MICROCONTROLADORES

PROGRAMACIÓN DE MICROCONTROLADORES PIC EN LENGUAJE C

Cándido Bariáin · Jesús M. Corres · Carlos Ruiz



Programación de microcontroladores PIC en lenguaje C

Programación de microcontroladores PIC en lenguaje C

Cándido Bariáin

Jesús María Corres

Carlos Ruiz

Marcombo

Programación de microcontroladores PIC en lenguaje C

Primera edición, 2017

Diseño de la cubierta: NDENU DISSENY GRÀFIC

ISBN: 978-84-267-2427-4
DL: B-25226-2016

Impreso en Ulzama Digital
Printed in Spain

PROGRAMACIÓN DE MICROCONTROLADORES PIC EN 'C'

ÍNDICE

ÍNDICE DE FIGURAS

Introducción

Este es un libro que guía en el aprendizaje de la programación en C de sistemas embebidos. C es un lenguaje de nivel medio, muy bien estructurado, que soporta el uso de funciones y módulos, pero que permite el uso de todas las características de bajo nivel, propias del ensamblador. Como el lenguaje C es eficiente y está muy probado en la programación de sistemas embebidos, es el más usado, por delante del ensamblador y de lenguajes de alto nivel orientados a objetos como C++ o Java. Los campos de aplicación son muchos: audio, automoción, comunicaciones, periféricos de ordenadores, equipos de consumo, instrumentación industrial, procesamiento de imagen, control de motores, robótica, etc. En este libro se usa el compilador de C de microchip y el microprocesador PIC16F877A, pero los conceptos y ejercicios son válidos para otras plataformas y compiladores mediante ligeras modificaciones.

El libro se ha estructurado para que el aprendizaje sea gradual: primero se introducen los esquemas de programación en C y después se aplican a los proyectos de diseño con el microcontrolador PIC. A lo largo de los distintos temas se han creado funciones que permiten un uso sencillo de los periféricos de los que dispone el microcontrolador. De esta manera la programación del microcontrolador para lograr que realice las tareas deseadas es más directa y el código, más fácil de interpretar.

El uso del simulador permite desarrollar programas sin necesidad de disponer de equipamiento *hardware*. Muchos de los ejercicios se pueden simular para comprobar su funcionamiento, lo que ahorra mucho tiempo en el proceso de diseño.

Bibliografía

- Hoja de Especificaciones del PIC16F877A. PIC16F87xA DataSheet. Microchip DS39582b (www.microchip.com).
- Guía de usuario HI-TECH C® for PIC10/12/16. DS51865A (www.microchip.com).
- Guía de usuario de los microcontroladores PIC de gama media. PICmicro Mid-Range MCU Family Reference Manual. DS33023a (www.microchip.com).
- Compiled Tips 'N Tricks Guide. DS01146B (www.microchip.com).
- Sensor de luminosidad TSL251RD. (www.ams.com).
- Sensor de temperatura digital TC74. DS21462D. (www.microchip.com).
- Memoria EEPROM 24LC256. DS21203M. (www.microchip.com).
- Sensor de temperatura TC1047A. DS21498D (www.microchip.com).
- Hoja de especificaciones del MAX232. (www.ti.com).
- Hoja de especificaciones de la pantalla LCD Hitachi HD44780 https://www.sparkfun.com/datasheets/LCD/HD44780.pdf.
- *The C Programming Language*, second edition, de Brian W. Kernighan y Dennis M. Ritchie. Ed. Prentice Hall.
- *Ejercicios de programación con microcontroladores PIC* de Jesús M. Corres, Carlos Ruiz y, Cándido Bariáin. Ed. Marcombo.

1. Introducción a la programación en C

1.1 Introducción

La programación en C de microcontroladores tiene importantes ventajas respecto al lenguaje ensamblador. C es un lenguaje de programación de propósito general, mientras que el lenguaje ensamblador está ligado a un microprocesador específico. Una vez se aprende a programar en C con un microcontrolador es muy sencillo cambiar a otra familia y, además, los códigos son mucho más fáciles de entender y modificar.

En este libro se emplean ejemplos cuya dificultad aumenta de forma progresiva para facilitar la comprensión de las estructuras de datos y de programación. Para profundizar más en el conocimiento de la programación en C es recomendable consultar libros de referencia como *The C Programming Language, second edition*, de Kernighan y Ritchie.

1.1.1 Estructura de un programa en C

Un programa en C contiene los siguientes elementos:

- **Directivas:** Las directivas del procesador le indican al compilador, por ejemplo, las definiciones de las etiquetas creadas por el programador, o los archivos externos que debe incluir.
- **Declaraciones:** En C se deben declarar las variables y las funciones antes de usarse por primera vez. En la declaración se indica el tipo de dato que almacenará la variable. Las variables globales se declaran fuera de las funciones y son visibles a partir del punto donde se realiza la declaración, mientras que las variables locales, que se declaran dentro de las funciones, solo son visibles dentro de la función.
- **Definiciones:** Establece el contenido de la variable o de la función.
- **Sentencias:** Todas las sentencias se terminan con ";".
- **Funciones:** Subprogramas que realizan una tarea concreta. Se delimitan mediante llaves {}. La función "main" es la primera en ejecutarse, indicando el principio y el fin del programa.

En el siguiente ejemplo de un programa en C se pueden observar sus distintos componentes:

<u>Código fuente</u>

```
//Directivas del preprocesador
#include <htc.h>
#include <stdio.h>
#include <math.h>
__CONFIG(FOSC_HS & WDTE_OFF & LVP_OFF & PWRTE_ON);  //Bits de configuración para
                                                   //el PIC
#define _XTAL_FREQ 20000000            //Frecuencia de trabajo a 20MHz

//Declaración y definición de variables globales
volatile signed char vector[90];

void main(void){              //Función main(), que se ejecuta tras un RESET

//Declaración de variables locales
      float x;
      int x1=-10;
      unsigned int x2=123;
      signed char x3=-10;
      char x4=200;
      float y=1.25;
      char z[20]="Cadena de caracteres";
```

```
      char i;

      printf("Esto es una prueba\n");
      printf("x1=%d\n",x1);
      printf("x2=%d\n",x2);
      printf("x3=%d\n",x3);
      printf("x4=%u\n",x4);
      printf("z=%s\n",z);
      x=0.0034;
      printf("x=%f\n",x);

      while(1);
}

void putch(unsigned char byte){          //Función putch(), que se ejecuta al
                                         //emplear printf
    TXSTA=0x26;
    RCSTA|=0x80;
    TXREG=byte;
    while(!TXIF)continue;
    TXIF=0;
}
```

Resultado:

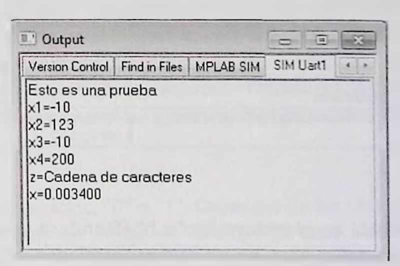


FIGURA 1.1: SIMULACIÓN EJEMPLO 1.1.1.

Al realizar un programa en C es necesario tener en cuenta que existe una serie de palabras reservadas que no se pueden utilizar para definir variables o nombres de funciones. A continuación, se muestra la lista de palabras reservadas de ANSI C:

```
auto        double      int         struct
break       else        long        switch
case        enum        register    typedef
char        extern      return      union
const       float       short       unsigned
continue    for         signed      void
default     goto        sizeof      volatile
do          if          static      while
```

1.1.2 Directivas del preprocesador

Las directivas aportan al compilador la información necesaria sobre el sistema (tipo de microcontrolador, oscilador, etc.).

La directiva #include permite insertar archivos completos. Estos archivos incluyen normalmente las definiciones de los registros del microcontrolador y las estructuras necesarias para acceder a cada uno de los bits de los mismos. Cuando el archivo se especifica con < > indica al compilador que lo busque en los directorios especificados, mientras que si se especifica "" le indica que lo haga primero en el directorio actual.

Ejemplo: #include <htc.h>

En la siguiente tabla se muestran las directivas de tipo "pragma", las cuales modifican el comportamiento del compilador.

Directive	Meaning	Example
inline	Specify function as inline	#pragma inline(fabs)
jis	Enable JIS character handling in strings	#pragma jis
nojis	Disable JIS character handling (default)	#pragma nojis
pack	Specify structure packing	#pragma pack 1
printf_check	Enable printf-style format string checking	#pragma printf_check(printf) const
regsused	Specify registers used by function	#pragma regsused wreg,fsr
switch	Specify code generation for switch statements	#pragma switch direct
warning	Control messaging parameters	#pragma warning disable 299,407

1.1.3. Función Printf

La función printf está definida en el archivo "stdio.h" (Standard Input Output). En un PC, la salida por defecto es la pantalla, pero en un microcontrolador es necesario especificar el tipo de dispositivo a emplear (pantalla LCD, RS232, etc.). La función putch permite especificar el tipo de salida. En el ejemplo de la sección 1.1.1 se usa el puerto serie (UART) del microcontrolador cuyo funcionamiento se detalla en el capítulo 6.

La entrada de la función printf consta de una cadena de control y una lista de argumentos:

```
printf ("cadena_de_control", argumentos);
```

Dentro de la cadena de control se pueden incluir caracteres constantes y códigos especiales. Los códigos especiales van precedidos por % y están vinculados a la siguiente lista de argumentos:

```
%c Caracter
%d entero decimal con signo
%f coma flotante notación decimal
%e coma flotante notación exponencial
%u entero decimal sin signo
%x entero sin signo hexadecimal minúsculas
%X entero sin signo hexadecimal mayúsculas
l prefijo usado junto a %d, %u, %x para especificar entero largo
```

```
%0 [width] Se imprimen los ceros a la izquierda. [width] indica el número total de
dígitos.
%[width].[precision] La precisión indica el número de decimales.
```

Códigos de escape:

```
\n nueva línea
\t tabulador horizontal
\r retorno de carro
\f formfeed
\' Comilla simple
\" Comilla doble
\\ Contrabarra
%% Símbolo de porcentaje
\? Interrogante
\b backspace
\0 Caracter nulo
\v Tabulador vertical
\xhhh Código hexadecimal hhh
```

1.1.4. Variables y tipos de datos

Una variable especifica una posición de memoria. Las posiciones de memoria pueden albergar distintos tipos de datos, dependiendo de cómo sea declarada. Como regla general se estudiará el tipo de dato que albergará la variable para asignarle el tipo de variable que mejor se ajuste economizando así la utilización de la memoria de datos, un recurso muy valioso y limitado cuando se trabaja con microcontroladores. A continuación, se detallan los diferentes tipos de datos y se muestran ejemplos de utilización de los mismos.

Bit

Un bit tiene dos posibles estados: "0" o "1". Cada uno de los LED o interruptores conectados a un puerto se pueden controlar mediante este tipo de dato.

Ejemplo 1.1.4A:

```
static bit b1;
b1=0;
printf("%d - %d \n", b1, ~b1);
```

Resultado:

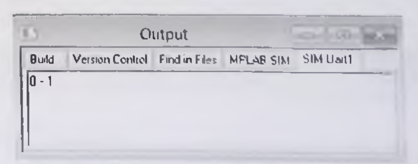


FIGURA 1.2: SIMULACIÓN EJEMPLO 1.1.4A.

Char

Los datos en la memoria del PIC se manejan como conjuntos de 8 bits. Si se considera que el número binario no tiene signo, el rango es de 0 a 255, mientras que si tiene signo es −128 a

127. Para indicarlo existen en C los modificadores *signed* y *unsigned*. Por ejemplo, para declarar las variables *x* e *y* como char con signo y sin signo respectivamente:

```
signed char x;
unsigned char y;
```

Int

Los datos enteros (integer) son tipos de datos de 16 bits. También pueden ser con signo o sin signo.

Ejemplo 1.1.4B:

```
int x1=-10000;
unsigned int x2=12300;
printf("x1=%d\n",x1);
printf("x2=%d\n",x2);
```

Resultado:

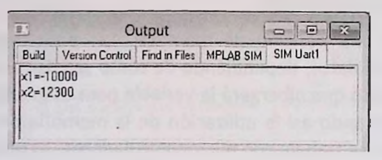


FIGURA 1.3: SIMULACIÓN EJEMPLO 1.1.4B.

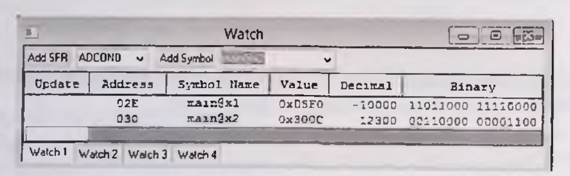


FIGURA 1.4: SIMULACIÓN EJEMPLO 1.1.4B, VALOR DE LAS VARIABLES.

Long

Son datos enteros de 32 bits.

Ejemplo: 1.1.4C

```
long unsigned int x2=123045;
printf("x2=%ld\n",x2);
```

Resultado:

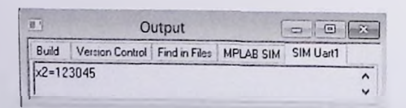


FIGURA 1.5: SIMULACIÓN EJEMPLO 1.1.4C.

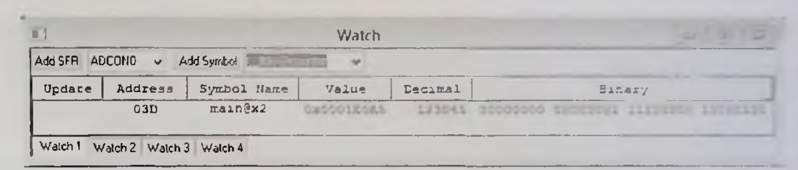


FIGURA 1.6: SIMULACIÓN EJEMPLO 1.1.4C, VALOR DE LAS VARIABLES.

Float, Double

Son números reales de 24 bits por defecto en formato IEEE754. Se pueden cambiar a 32 bits (Project/Build options/Project/Global).

Format	Number	biased exponent	1.mantissa	decimal
32-bit	7DA6B69Bh	11111011b	1.01001101011011010011011b	2.77000e+37
		(251)	(1.302447676659)	
24-bit	42123Ah	10000100b	1.001001000111010b	36.557
		(132)	(1.142395019531)	

Ejemplo 1.1.4D:

```
float y=1.25, x=0.0034;
printf("x=%f\n",x);
```

Resultado:

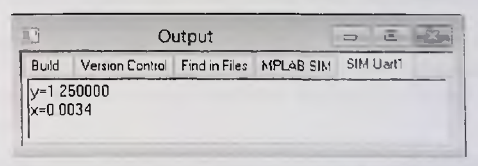


FIGURA 1.7: SIMULACIÓN EJEMPLO 1.1.4D.

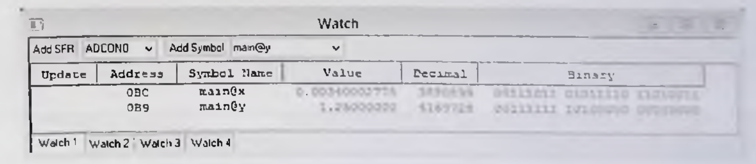


FIGURA 1.8: SIMULACIÓN EJEMPLO 1.1.4D, VALOR DE LAS VARIABLES.

1.1.5 Definiciones de constantes

Las definiciones de constantes no se almacenan en memoria, se resuelven en el momento de la compilación y para ello se usa la directiva ***define***.

```
#define <label> value
```

Por ejemplo:

```
#define pi 3.14159265359
```

Para almacenar constantes en memoria ROM, se debe usar el modificador ***const***

```
char const id[5]={"1234"};
printf("%s\n",id);
```

Resultado:

FIGURA 1.9: SIMULACIÓN EJEMPLO 1.1.5.

1.1.6 Macros y compilación condicional

Usando la directiva ***#define*** también se pueden implementar macros:

```
#define MAX(A,B) (A>B)?A:B
z=MAX(x,y); // z contiene el valor mayor de x e y
```

Para la compilación condicional de secciones del programa se emplean las siguientes directivas del preprocesador: #define, #if, #endif, #else

Ejemplo:

```
#define output_high(A) A=1
#define HW_VERSION 5
#if HW_VERSION>3
output_high(RB0);
#else
output_high(RB1);
#endif
```

1.2. Uso de las variables

1.2.1 Declaraciones

Las variables se pueden declarar dentro o fuera de las funciones, según sean locales o globales respectivamente.

Las *variables locales* definidas en distintas funciones pueden compartir el mismo nombre y solo existirán durante la ejecución de la función y se liberarán posteriormente para su reutilización por otras funciones excepto en el caso de ser de tipo ***static*** (véase apartado 1.2.3).

```
void f2(void){
	int count;
	for (count = 0 ; count < 10 ; count++)
	printf("%d ",count);
}
f1(){
	int count;
	for (count=0; count<10; count++){
	  f2();
	  printf(" - %d \n",count);
	}
}

void main(void){
	f1();
	while(1);
}
```

Resultado:

FIGURA 1.10: SIMULACIÓN EJEMPLO 1.2.1.

Las *variables globales* se pueden usar en distintas funciones, y deben declararse antes de usarse por primera vez.

```
int max;
f1(){
	int i;
	for(i=0; i<max;i++)
	printf("%d ",i);
}

void main(void){
	max=10;
	f1();
	while(1);
}
```

Mediante ***typedef*** se pueden definir nuevos tipos en función de tipos ya existentes:

```
typedef signed char int8;
int8 i=12;
```

La palabra reservada ***enum*** permite enumerar automáticamente cualquier lista de identificadores que se le pase, asignándoles valores de 0, 1, 2, etc. Un tipo de datos

enumerado es una manera de asociar nombres a números y, por consiguiente, de ofrecer más significado a alguien que lea el código. Se pueden definir así nuevos tipos de variables mediante *enum* (que se representan siempre como valores enteros). La declaración de un tipo de datos enumerado se parece a la declaración de un ***struct*** (apartado 1.8.1)

```
enum nombre_tipo{
      Nombre1,
      Nombre2,
      …
      NombreN
};
```

Ejemplo 1.2.1B:

```
enum semana{Lunes, Martes, Miercoles, Jueves, Viernes, Sabado, Domingo};
void main(void){
      enum semana actual=Jueves;
      switch(actual) {
         case Lunes:  printf("Lunes"); break;
         case Martes:  printf("Martes"); break;
         case Miercoles:  printf("Miercoles"); break;
         case Jueves:  printf("Jueves"); break;
         case Viernes:  printf("Viernes"); break;
         case Sabado:  printf("Sabado"); break;
         case Domingo:  printf("Domingo"); break;
      }
}
```

Resultado:

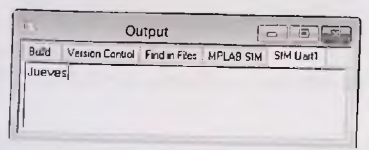


FIGURA 1.11: SIMULACIÓN EJEMPLO 1.2.1B.

1.2.2 Conversiones de tipos

Como C permite mezclar distintos tipos de datos en la misma expresión, tiene unas reglas de promoción de tipos para convertir las variables al tipo de la variable de mayor tamaño.

En el siguiente ejemplo:

```
char ch = '0';
int i = 15;
float f = 25.6;
double result;
result= ch*i/f;
```

- Primero la variable ch de tipo char (8 bits) se convierte a entero (16 bits).
- Se multiplica ch * i

- Se convierte ch*i a float (24/32 bits)
- Se divide entre f
- El float se convierte a double (24/32 bits) y se almacena en result (24/32 bits).

También se puede forzar la conversión de tipos usando el siguiente formato: (tipo) valor
En el siguiente ejemplo se convierte el float en int:

```
float f;
f = 100.2;
printf("%d \n",(int)f);
printf("%d \n",f);
```

Obsérvese que el resultado del segundo printf no es correcto, mientras que el primero sí.

Resultado:

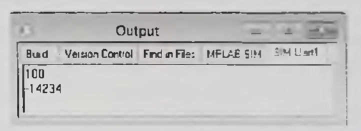


FIGURA 1.12: SIMULACIÓN EJEMPLO 1.2.2.

1.2.3 Clases de almacenamiento

Hay cuatro clases de almacenamiento en C: ***auto***, ***extern***, ***static*** y ***register***. Este último no se usa en el PIC.

auto: Las variables definidas dentro de una función son ***auto*** por defecto. El compilador asigna un bloque de RAM a esas variables y las posiciones de memoria que ocupan se pueden reutilizar cuando se sale de la función.

extern: La variable declarada como ***extern*** se define en otro fichero.

static: Las variables definidas como ***static*** son *globalmente activas* y solo se inicializan una vez, aunque se definan dentro de una función.

En el siguiente ejemplo:

```
void test(){
      char x,y,z;
      static int count = 4;
      printf("count = %d\n",++count);
}
```

La variable count se incrementa con cada llamada de la función test: 4, 5, 6...

No es una variable global, porque no se puede acceder a ella desde otras funciones.

1.3. Funciones

Las funciones son los bloques con los que se construyen los programas en C. Encapsulan partes del código, ayudando a que se puedan reutilizar en otros programas.

1.3.1. Paso de argumentos a las funciones

Es posible pasar hasta 31 argumentos a una función. Si la función no se ha definido previamente a su uso por primera vez, al menos es necesario declararla. Cuando no hay argumentos o no se devuelve ningún resultado se usa ***void***.

Ejemplo:

```
void suma(int a, int b);           //Declaración del prototipo de la función
void resta(int a, int b);          //Declaración del prototipo de la función

void main(void){
      int a=10,b=1;
      suma(a,b);
      resta(a,b);
      printf("Operandos: %d,%d\n",a,b);
}

void suma(int a, int b){
      a=a+b;
      printf("Suma: %d\n",a);
}
void resta(int a, int b){
      b=a-b;
      printf("Resta: %d\n",b);
}
```

En el ejemplo anterior se han pasado los argumentos por valor. Cualquier cambio realizado en la variable no modifica el valor original pasado como argumento como se muestra a continuación.

Resultado:

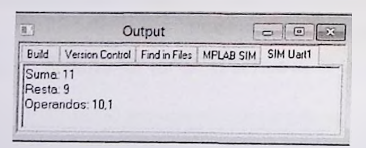


FIGURA 1.13: SIMULACIÓN EJEMPLO 1.3.1.

También se puede pasar el argumento utilizando el "*paso por referencia*". En este caso, se pasa la dirección de la variable, con lo que es posible *modificar el valor de la variable original* dentro de la función.

En el siguiente ejemplo se pasa la dirección de la variable "input1" a la función neg, es decir, su puntero. Esta función calcula el opuesto y modifica el valor de la variable cuyo puntero se ha pasado.

```
void neg(int *input);                //Declaración del prototipo de la función

void main(void){
      int input1=10;
      neg(&input1);
      printf("Variable: %d\n",input1);
      while(1);
}

void neg(int *input) {
      *input=-*input;
}
```

Resultado:

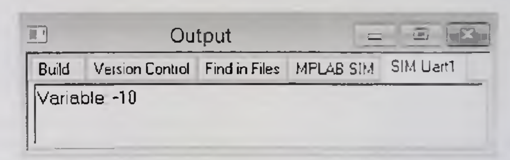


FIGURA 1.13: SIMULACIÓN EJEMPLO 1.3.13.

1.3.2 Retorno de resultados

Para hacer que una función devuelva un resultado se usa **return**.

```
int func();                          //Declaración del prototipo de la función
int sum(int a, int b);               //Declaración del prototipo de la función

void main(void){
      int num;
      num = func();
      printf("%d\n", num);
      num = sum(5,127);
      printf("%d\n",num);
      while(1);
}

int func(){
      return 6;
}
int sum(int a, int b){
      int result;
      result = a + b;
      return result;
}
```

En cuanto una función llega a la línea en que está el comando ***return***, termina su ejecución. Todas las sentencias que se encuentren después no se ejecutarán.

1.3.3 Interrupciones

Son funciones especiales que se activan por un evento *hardware*. Se declaran mediante el modificador ***interrupt***.

Las interrupciones deben ser habilitadas en el programa principal.

En el siguiente ejemplo la interrupción ISR se ejecuta cada vez que el temporizador TMR0 se desborda (el funcionamiento de los temporizadores se describirá en el Capítulo 3), incrementando el contador x1.

```
#include <htc.h>
#include <stdio.h>
#include "interrupts.h"
#include "timers.h"
__CONFIG(FOSC_HS & WDTE_OFF & LVP_OFF & PWRTE_ON);
#define _XTAL_FREQ 20000000

void interrupt ISR(void);          //Declaración del prototipo de la función

int x1=0;                          //Declaración de variable global

void main(void){
      ei();                               //Habilita interrupciones globales
      enable_interrupts(INT_T0);          //Habilita interrupción del Timer 0
      setup_timer0(RTCC_INTERNAL|RTCC_DIV2);  //Configura el Timer 0
      while(1);
}

void interrupt ISR(void){
      if(INTCONbits.TMR0IF == 1 && INTCONbits.TMR0IE == 1){
        set_timer0(0x00);                     //Ajusta el contador del TIMER0
        INTCONbits.TMR0IF = 0;
        printf("%d\n",x1);
        x1++;
      }
}

void putch(unsigned char byte){
      TXSTA=0x26;
      RCSTA|=0x80;
      TXREG=byte;
      while(!TXIF)continue;
      TXIF=0;
}
```

Resultado:

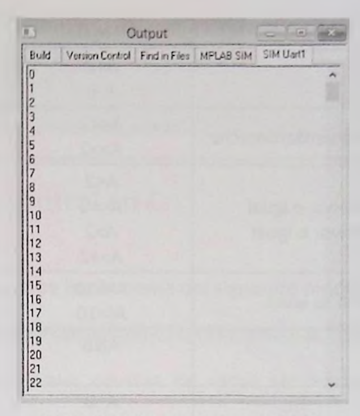


FIGURA 1.14: SIMULACIÓN EJEMPLO 1.3.3.

1.4. Operadores

En la siguiente tabla se muestran los operadores C en orden de precedencia, de mayor a menor. Su asociatividad indica cómo se aplican en una expresión los operadores de la misma precedencia.

En la cuarta columna se muestra el resultado de la operación para una variable A = 10 y otra variable B = 20.

Operador	Descripción	Ejemplo	Resultado	Asociatividad
( ) [ ] . -> ++ --	Paso de parámetros a funciones Índices en arrays Selección de elemento mediante objeto Selección de elemento mediante puntero Postincremento Postdecremento	 A++ A--	 11 9	Izquierda a derecha
++ -- + - ! ~ (*type*) * & sizeof	Preincremento Predecremento + / – unitario Negación lógica / complement bit a bit Conversión de tipos de datos Referencia al valor Operador de dirección de memoria Valor en bytes de la variable	++A --A	11 9	Derecha a izquierda
* / %	Multiplicación/división/módulo	A*B A/B A%B	200 2 0	Izquierda a derecha

Operador	Descripción	Ejemplo	Resultado	Asociatividad
+ -	Suma/Resta	A+B A-B	30 -10	Izquierda a derecha
<< >>	Rotación de bits a izquierda/derecha	A<<2 A>>2		Izquierda a derecha
< <= > >=	Relacional menor/menor o igual Relacional mayor/mayor o igual	A<2 A<=2 A>2 A>=2	0 0 1 1	Izquierda a derecha
== !=	Relacional es igual/no es igual	A==10 A!=10	1 0	Izquierda a derecha
&	AND bit a bit	A&B	0	Izquierda a derecha
^	XOR bit a bit	A^B	30	Izquierda a derecha
|	OR bit a bit	A|B	30	Izquierda a derecha
&&	AND lógico	A&&B	0	Izquierda a derecha
||	OR lógico	A||B	1	Izquierda a derecha
? :	Condicional ternaria			Derecha a izquierda
= += -= *= /= %= &= ^= |= <<= >>=	Asignación Suma/resta y asignación Multiplicación/división y asignación Módulo / And y asignación Xor / Or y asignación Desplazamiento izquierda/derecha y asig.			Derecha a izquierda
,	Coma (separa expresiones)			Izquierda a derecha

Notas:

- Los paréntesis se usan para agrupar subexpresiones forzando una precedencia diferente. Se evalúan de dentro hacia afuera.
- Es importante no confundir la asignación "=" con el operador relacional "==".
- Los operadores ++ y -- siempre incrementan o decrementan una unidad. Cuando preceden a la variable, esta se incrementa/decrementa y después se usa en la expresión. Cuando siguen a la variable, esta se usa en la expresión y después se incrementa/decrementa.

Ejemplos:

```
sum = a+b++       ->    sum = a+b b = b+1
sum = a+b--       ->    sum = a+b b = b-1
sum = a+ ++b      ->    b = b+1 sum = a+b
sum = a+ --b      ->    b = b-1 sum = a+b
```

1.5. Estructuras de control en C

1.5.1 Condicionales

La estructura condicional en C se implementa del siguiente modo:

```
if (expresión) sentencia; else sentencia_alternativa;
```

Cuando los bloques condicionales constan de varias sentencias, estas se agrupan entre llaves:

```
if (expresión) {sentencia1; sentencia2; ...} else {sentencia_alternativa1; ...}
```

Ejemplo:

```
if (count <0) printf("Negativo\n");
else if (count ==0) printf("Cero\n");
else printf("Positivo\n");
```

También se puede expresar del siguiente modo:

```
expresión1 ? expresión2 : expresión3
```

Primero se evalúa la expresión1. Si es verdadera, entonces se evalúa la expresión2, y si es falsa, la expresión3.

Ejemplo:

```
i ? j=0 : j=1;
```

1.5.2 Bucles de tipo FOR

La estructura FOR se implementa en C de la siguiente manera:

```
for (sentencia_inicialización ; sentencia_condición ; incremento)
{sentencia1; sentencia2; ...}
```

La *sentencia de inicialización* da el valor inicial a las variables que se usan en el bucle y se ejecuta una vez.

La *sentencia de condición* se evalúa antes de cada iteración y determina si se ejecuta o no el bucle.

Con cada iteración se ejecuta el *incremento* de la variable de control.

```
void main(void){
      int i;
      for(i=0; i<10; i++) printf("%d ",i);
}
```

Resultado:

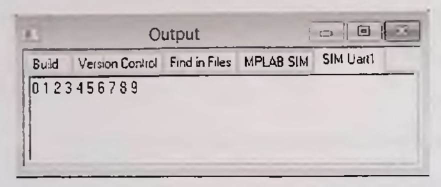


FIGURA 1.15: SIMULACIÓN EJEMPLO 1.5.2.

1.5.3 Bucles de tipo WHILE

```
while (expresión) {sentencial; ...}
```

Primero se evalúa la expresión y si es verdadera se ejecuta una iteración del bucle.

Ejemplo:

```
void main(void){
      int i=0;
      while(i<10)      printf("%d ",i++);
}
```

Resultado:

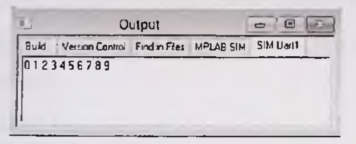


FIGURA 1.16: SIMULACIÓN EJEMPLO 1.5.3.

1.5.4 Bucles de tipo DO-WHILE

```
do {sentencial; sentencia2; ...} while (expresión)
```

Primero se ejecutan las sentencias y después se evalúa la expresión. Si es falsa se termina el bucle.

```
void main(void){
      int i=0;
      do printf("%d ",i++); while (i<10);
}
```

Resultado:

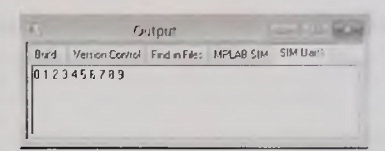


FIGURA 1.17: SIMULACIÓN EJEMPLO 1.5.4.

1.5.5 Break

Cuando un bucle encuentra una sentencia ***break*** se finaliza el bucle en el que se encuentre (if, for, while, etc.)

1.5.6 Continue

Cuando un bucle encuentra una sentencia **continue** se salta las sentencias que le siguen hasta la siguiente condición de test.

```
void main(void){
      int i;
      for(i=0;i<10;i++){
        if (i<5){
               printf("x");
               continue;
        }
        printf("%d ",i);
      }
}
```

Resultado:

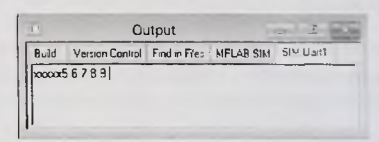


FIGURA 1.18: SIMULACIÓN EJEMPLO 1.5.6.

1.5.7 Condicionales switch - case

```
switch (variable)
{
      case constant1:
        statement(s);
        break;
      case constant2:
        statement(s);
        break;
      case constantN:
        statement(s);
        break;
      default:
        statement(s);
}
```

Se va comprobando la variable con cada una de las constantes. ***Break*** y ***default*** son opcionales.

Ejemplo:

```
bit b=0;
switch (b){
      case 0:    printf("b is false");
                 break;
      case 1:    printf("b is true");
                 break;
}
```

1.6. Vectores

1.6.1 Vectores unidimensionales

La forma general de un vector unidimensional es:

```
type var_name [size];
```

El índice de los elementos va desde 0 hasta size-1.
Por ejemplo `int height[50];` declara un vector de 50 enteros.
Para asignar el valor 10 al primer elemento y al último usamos `height[0]=10; height[49]=10;`
C asigna los elementos de un vector en posiciones contiguas de la memoria.
Para inicializar el vector se usan las llaves `int height[50]={1,2,3,4,5,...};`

1.6.2. Cadenas de caracteres

Una cadena es un vector de tipos *char* que termina con el carácter nulo \0. La librería string.h incluye funciones que permiten trabajar con este tipo de estructuras de forma sencilla como se muestra en el siguiente ejemplo que crea una cadena de caracteres.

Código fuente

```
#include <string.h>
void main(void){
      char str[10];
      strcpy(str, "Esto es una prueba");
      printf("%s",str);
}
```

Resultado:

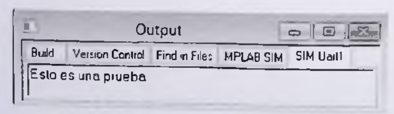


FIGURA 1.19: SIMULACIÓN EJEMPLO 1.6.2.

Funciones para manejo de cadenas en la librería string.h:

```
strcat Concatena dos cadenas
strchr Busca el primer carácter dentro de una cadena
strrchr Busca el último carácter dentro de una cadena
strcmp Compara dos cadenas
strncmp Compara un número de caracteres dentro de una cadena
stricmp Compara dos cadenas ignorando la diferencia entre mayúsculas y minúsculas
strncpy Copia un número de caracteres de una cadena en otra
strlen Calcula la longitud de una cadena
strlwr Reemplaza mayúsculas con minúsculas
strpbrk Localiza el primer carácter que coincide en dos cadenas
strstr Localiza la primera ocurrencia de una subcadena de caracteres dentro de
una cadena
```

Realizar programas que permitan comprobar el funcionamiento de las funciones de la librería para el manejo de cadenas de caracteres "string.h".

1.6.3 Vectores multidimensionales

La forma general de un vector multidimensional es:

```
type var_name [size1][size2][...][sizeN];
```

El índice de los elementos va desde 0 hasta sizeX-1.

Por ejemplo `int height[2][2];` declara una matriz de 2 filas por 2 columnas (4 enteros).

Para asignar el valor 10 al elemento de la fila 2 columna 1 utilizamos `height[1][0]=10;`

C asigna los elementos de un vector en posiciones contiguas de la memoria.

Para inicializar el vector se usan las llaves `int height[2][2]={1,2,3,4};`

Ejemplo:

```
void main(void){
      int array[5][4];
      int i,j;
      for(i=0;i<5;i++) for(j=0;j<4;j++) array[i][j]=i*j;

      for(i=0;i<5;i++){
         for(j=0;j<4;j++) printf("%02d ",array[i][j]);
         printf("\n");
      }
}
```

Resultado:

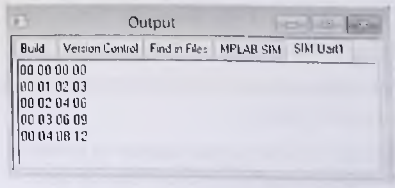


FIGURA 1.20: SIMULACIÓN EJEMPLO 1.6.3.

1.7. Punteros

Un puntero es una variable que contiene la dirección de memoria de otra variable.

La forma de declarar un puntero es mediante el asterisco:

```
tipo *variable
```

donde *tipo* especifica el tipo de variables a las que la variable apunta.

Por ejemplo, para declarar un puntero de nombre *ptr* a variables de tipo entero se usa `int *ptr;`

Una vez declarado, para acceder a la dirección de una variable usamos el símbolo &, mientras que el operador * devuelve el valor almacenado en la dirección apuntada por la variable.

Ejemplo: Se declara un puntero entero *a*, y un entero *b*. Se asigna al entero *b* el valor 6 y al puntero *a* la dirección del entero *b*. Al imprimir el valor apuntado por *a*, nos estamos dirigiendo al valor de la variable entera *b*, que es 6.

Código fuente A

```
void main(void){
      int *a,b;
      b=6;
      a=&b;
      printf("%d",*a);
}
```

Resultado:

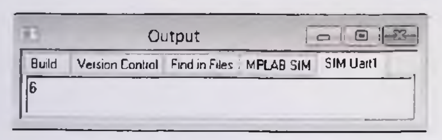


FIGURA 1.21: SIMULACIÓN EJEMPLO 1.7ª.

Los punteros se pueden incrementar y decrementar, lo que es muy útil para acceder a elementos dentro de vectores o tablas.

Por ejemplo, al escribir *a*++, dependiendo del tipo de dato al que apunta, al incrementarse se suma 1 si es char, 2 si es entero, etc.

También se puede incrementar el valor al que apunta el puntero: (*a)++. En este caso se incrementa en una unidad, independientemente del tipo de dato.

En el caso de datos de tipo vector, el nombre del vector es un puntero al primer elemento del mismo.

En el siguiente ejemplo, ambas instrucciones printf muestran el mismo resultado.

Código fuente B

```
void main(void){
	int a[5]={1,2,3,4,5};
	int *p;
	p=a;
	printf("%d \n",*(p+3)); printf("%d \n",a[3]);
}
```

Resultado:

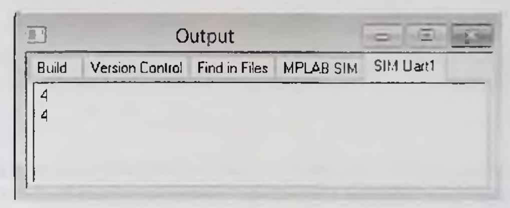


FIGURA 1.22: SIMULACIÓN EJEMPLO 1.7B.

1.8. Estructuras y Uniones

1.8.1. Estructuras

Las estructuras sirven para agrupar distintos datos dentro de un mismo contenedor, para poder acceder a ellos de forma más sencilla, mediante un nombre de objeto común. Dentro de la estructura, cada elemento puede tener su propio tipo de dato.

```
struct etiqueta{
	type elemento1;
	type elemento2;
	...
	type elementon;
} lista de variables;
```

etiqueta es el nombre de la estructura mientras que *lista de variables* es la lista con los nombres de las variables declaradas de ese tipo de estructura.

En el siguiente ejemplo se declara una estructura llamada *card* de tipo *catalog*. que agrupa 5 campos de datos.

Ejemplo:

```
struct catalog{
	char author[40];
```

```
        char title[40];
        char pub[40];
        unsigned int data;
        unsigned char rev;
} card;
```

Para acceder a los campos de la variable card se utiliza el punto, que separa el nombre de la variable de los elementos que la componen:

```
card.rev='w';
card.data=12;
printf("%s",card.author);
```

Una vez definida la estructura se pueden declarar más variables del mismo tipo, incluso vectores de estructuras como en el caso de la variable *bigcat* que se muestra a continuación:

```
struct catalog card2,card3,bigcat[3];
bigcat[2].data=12;
```

1.8.2. Uniones

Una unión es una única posición de memoria que se comparte entre dos o más variables. Las uniones son útiles para ahorrar espacio de memoria y para acceder a la posición de memoria con diferentes formatos.

Para declarar una unión se procede de forma similar a una estructura:

```
union etiqueta{
        type elemento1;
        type elemento2;
        ...
        type elementon;
} lista de variables;
```

En el siguiente ejemplo se declara una variable *temp* que está formada por la unión de un vector de tres caracteres, un entero de 16 bits y un real de 32 bits.

```
union u_type{
        int i;
        char c[3];
        double d;
} temp;
```

double			
c[2]	c[1]	c[0]	
int			
elemento0	elemento1	elemento2	elemento3

El entero usa dos bytes, el vector de char usa 3 y el double usa los 4.

En el siguiente ejemplo se crea una variable s1 formada por la unión de un vector de 2 caracteres y un entero con signo.

```
union sample{
      unsigned char bytes[2];
      signed short word;
} s1
```

Utilizando la unión podemos acceder a los 16 bytes: `s1.word`; o a cada uno de los bytes independientemente: `s1.bytes[0]; s1.bytes[1];`



2. Puertos de entrada y salida digitales

2.1. LED y Pulsadores
2.2. Visualizadores numéricos de 7 segmentos
2.3. Exploración de teclado matricial
2.4. Control de Pantallas LCD

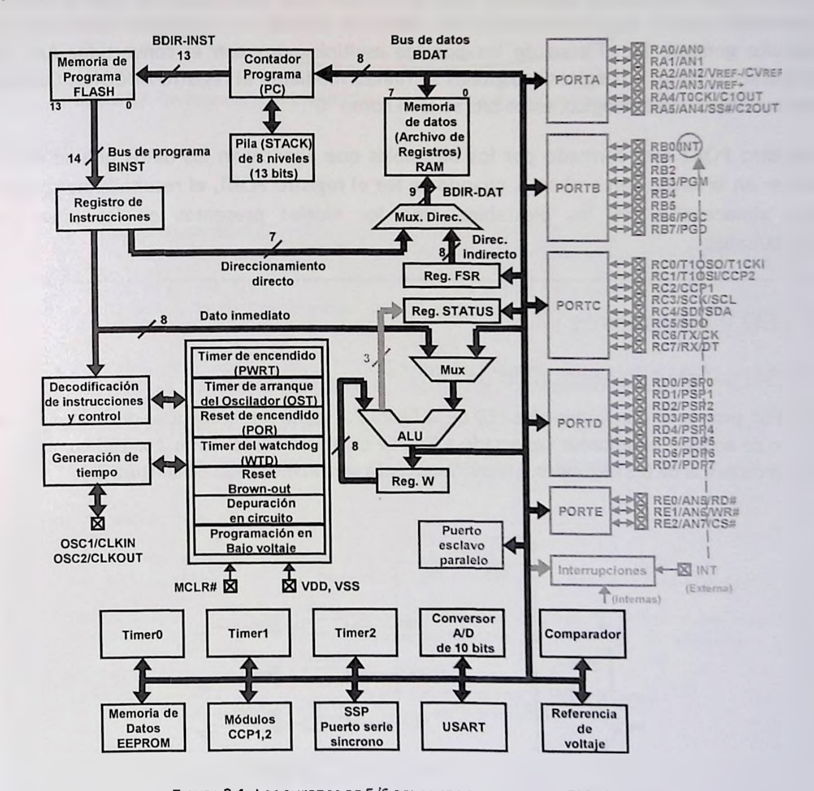


FIGURA 2.1: LOS PUERTOS DE E/S DEL MICROCONTROLADOR PIC16F877A.

Los puertos digitales de entrada y salida son los periféricos más sencillos de que dispone el microcontrolador, y mediante los cuales es posible controlar otros dispositivos y monitorizar su estado.

El PIC16F877A tiene los terminales de entrada/salida divididos en 5 puertos (desde el PORTA hasta el PORTE), algunos disponen de hasta 8 terminales. Los terminales se pueden

configurar como entrada o salida mediante el correspondiente registro de dirección de datos (TRIA, TRISB, etc). Los bits del registro TRIS que se escriben con "1" indican que los terminales correspondientes del PORT son entradas (Input), mientras que los se escriben con "0" indican que los terminales son de salida (Output). Es fácil de recordar ya que el "1" es similar a la "I" de "Input" y el "0" a la "O" de "Output".

Para aumentar la flexibilidad del PIC la mayoría de los pines están multiplexados con funciones alternativas, de modo que cuando un periférico como, por ejemplo, la USART (Transmisión Serie) está funcionando sus pines no podrán ser utilizados como I/O de propósito general. En el caso de los puertos multiplexados con el convertidor A/D, la configuración como analógicos o digitales se realiza mediante el registro ADCON1. Cuando se selecciona como analógico, estos bits se leen como "0".

El registro PORT está formado por los biestables que almacenan los datos que se van a mostrar en la salida. Sin embargo, cuando se lee el registro PORT, el resultado no son los datos almacenados en los biestables, sino los niveles presentes en los pines de entrada/salida.

2.1 LED y Pulsadores

2.1.1 Secuencias de encendido de LED

Realiza un programa que ilumine los LED de la figura 2.2 según dos secuencias diferentes. Cada vez que se accione el pulsador conectado a RA4 se cambiará la secuencia. Los LED cambiarán con una cadencia de 0.5 segundos. El funcionamiento será el mostrado en la figura 2.3.

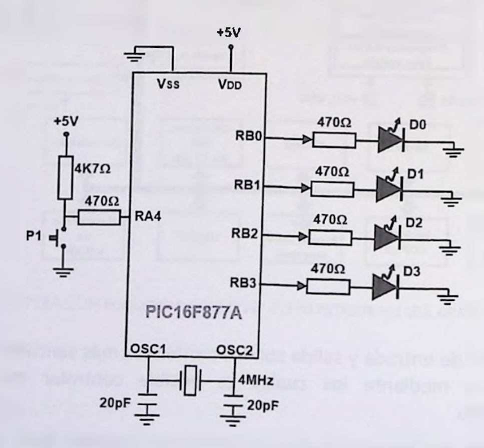


FIGURA 2.2: CONEXIONES CON EL PIC16F877A.

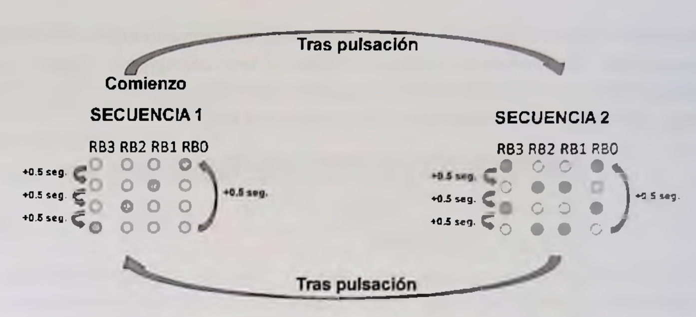


FIGURA 2.3: DIAGRAMA DE FUNCIONAMIENTO DE LA SECUENCIA DE ENCENDIDO/APAGADO DE LOS LED.

Solución:

Código fuente

```
//ARCHIVOS DE DEFINICIONES
#include <htc.h>                        //Incluimos librería del micro a usar
#include "ports.h"
#define _XTAL_FREQ 4000000              //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);   //Configuración
                                                                //del PIC
//Variables
bit CAMBIAR=0;                    //Flag para cambiar la secuencia de los LED
    //(SECUENCIA1 si CAMBIAR=0) y (SECUENCIA2 si CAMBIAR=1) - Empieza con la
    //SECUENCIA1
unsigned char SECUENCIA1=0x11;          //Secuencia 1 empieza con LED D0
                                        //encendido
unsigned char SECUENCIA2=0x09;          //Secuencia 2 empieza con LEDs D3-D0
                                        //encendidos

//PROGRAMA PRINCIPAL
void main()
{       //INICIALIZACIONES PARA EL PIC
        set_all_digital();      //Desactivamos PORTA como entradas analógicas

        TRISA=0x10;             //PORTA con RA4 como entrada y resto como salidas
        TRISB=0x00;             //PORTB con todos los terminales de salida.

        while(1)
        {  if(RA4 == 0){              //Se ha presionado el pulsador
                CAMBIAR=~CAMBIAR;     //Cambiar secuencia
           }
           if(CAMBIAR==0){
                PORTB=SECUENCIA1;
                __delay_ms(500);        //Esperar 500 ms
                rol_8 (SECUENCIA1,1);   //Rotar un bit a la izquierda en SECUENCIA1
           }
           else{
                PORTB=SECUENCIA2;
                __delay_ms(500);            //Esperar 500 ms
                SECUENCIA2=~SECUENCIA2;     //Cambiar LED encendidos en SECUENCIA2
           }
        }
}
```

```
//PROGRAMA PRINCIPAL
void main(
      set_all_digital();              //Desactivar PORTA como entradas analógicas
      TRISA=0x10;                     //Configurar RA4 como entrada
      TRISB=0x00;                     //Configurar RB0 como salida
      PORTB=0;                        //Todos los led apagados
      ESTADO=1;                       //Empieza con la señal activada

      while(1){
        if(ESTADO==1){                //Si señal activada
                punto();
                punto();
                punto();
                raya();
                raya();
                raya();
                punto();
                punto();
                punto();
                pausa();}
        else {PORTB=0;}               //Si señal desactivada todos los led apagados
      }
}
```

Comprueba el funcionamiento mediante la utilización del simulador MPLAB SIM y el visualizador de señales digitales (menú View⇒ Simulator Logic Analyzer) observando la salida RB0 y el generador de estímulos en modo asíncrono (menú Debugger⇒ Stimulus) asociado al terminal RA4.

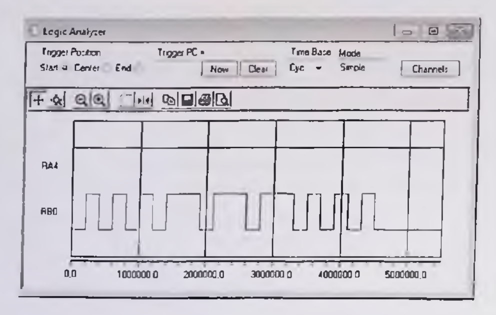


FIGURA 2.6: SIMULACIÓN DEL EJERCICIO 2.1.2.

2.1.3 Visualización de una cadena de caracteres en código morse mediante un LED

Escribe un programa para la tarjeta PICDEM2PLUS que genere el código morse de una cadena de caracteres y lo visualice en un LED conectado a la salida RB0 (Figura 2.5).

La duración de cada punto será de 10 ms y cada raya de 30 ms. El silencio entre símbolos (punto/raya) serán 10 ms, mientras que la separación entre caracteres serán 30 ms y entre palabras de 70 ms.

Solución:

La función morse() recorre la cadena de caracteres que se le pasa como parámetro de entrada, detectando el tipo de caracter y llamando a la función flashLED() con el código de puntos y barras que le corresponde. Esta función decodifica el código, el cual consiste en dos campos diferentes: el primero (los tres bits más significativos) indica el número de elementos (puntos y rayas) y el segundo el tipo de símbolo en cada posición, empezando por el bit menos significativo.

Código fuente

```
//ARCHIVOS DE DEFINICIONES
#include <htc.h>                    //Incluimos librería del micro a usar
#include "ports.h"
#define _XTAL_FREQ 4000000          //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);
void morse (char txtString[]);
// PROGRAMA PRINCIPAL
                                    //Duración de 1 elemento: 10 ms
                                    //Punto = 1 elemento
                                    //Raya = 3 elementos
                                    //Espacio entre símbolos = 1 elemento
                                    //Espacio entre caracteres = 3 elementos
                                    //Espacio entre palabras = 7 elementos
void main(){
      set_all_digital();            //Desactivar PORTA como entradas analógicas
      TRISB=0x00;                   //Configurar RB0 como salida
      RB0=0;                        //Todos los led apagados
      morse("Hola Mundo");
      while(1);
}
   //Definición de puntos y rayas
   //Para caracteres alfanuméricos, 3 MSB bits definen el número de símbolos (1 a 5)
   //Para caracteres de puntuación (',' y '.'), 2 MSB bits indican 6 símbolos
   //Punto:0, raya:1;
   //LSB bits almacenados en orden inverso de modo que el bit 0 = primer símbolo
   //Ejemplo, F = "..-." = 4 símbolos = 0010, por lo tanto se almacena como 0100
   //000 xxxxx 0 símbolos
   //001 xxxx? 1 símbolos x = no importa, ? = dato del símbolo = 0 ó 1)
   //010 xxx?? 2 símbolos
   //011 xx??? 3 símbolos
   //100 x???? 4 símbolos
   //101 ????? 5 símbolos
   //11 ?????? 6 símbolos
char Tabla[] =   {0b01000010,      //  0 = A ".-"
                      0b10000001,  //  1 = B "-..."
                      0b10000101,  //  2 = C "-.-."
                      0b01100001,  //  3 = D "-.."
                      0b00100000,  //  4 = E "."
                      0b10000100,  //  5 = F "..-."
                      0b01100011,  //  6 = G "--."
                      0b10000000,  //  7 = H "...."
                      0b01000000,  //  8 = I ".."
                      0b10001110,  //  9 = J ".---"
                      0b01100101,  // 10 = K "-.-"
                      0b10000010,  // 11 = L ".-.."
                      0b01000011,  // 12 = M "--"
                      0b01000001,  // 13 = N "-."
```

```
                        0b01100111, // 14 = O "---"
                        0b10000110, // 15 = P ".--."
                        0b10001011, // 16 = Q "--.-"
                        0b01100010, // 17 = R ".-."
                        0b01100000, // 18 = S "..."
                        0b00100001, // 19 = T "-"
                        0b01100100, // 20 = U "..-"
                        0b10001000, // 21 = V "...-"
                        0b01100110, // 22 = W ".--"
                        0b10001001, // 23 = X "-..-"
                        0b10001101, // 24 = Y "-.--"
                        0b10000011, // 25 = Z "--.."
                        0b10111111, // 26 = 0 "-----"
                        0b10111110, // 27 = 1 ".----"
                        0b10111100, // 28 = 2 "..---"
                        0b10111000, // 29 = 3 "...--"
                        0b10110000, // 30 = 4 "....-"
                        0b10100000, // 31 = 5 "....."
                        0b10100001, // 32 = 6 "-...."
                        0b10100011, // 33 = 7 "--..."
                        0b10100111, // 33 = 8 "---.."
                        0b10101111, // 34 = 9 "----."
                        0b11110011, // 36 = , "--..--"
                        0b11101010  // 37 = . ".-.-.-"
                        };
void flashLED (char puntuacion) {
  char count;
  count = (puntuacion >> 5) & 0b00000111;     //Extraer los tres bits superiores
                                               //de la cuenta
  if (count >= 6) count &= 0b00000110;         //Comprobar si es un carácter de
                                               //puntuación
  for (int i = 0; i < count; i++) {
      RB0=1;
      if ((puntuacion & 0b00000001) == 0)      //Punto
      __delay_ms(10);
      else                                     //Raya
      __delay_ms(30);
      puntuacion = puntuacion >> 1;
      RB0=0;
      __delay_ms(10);                          //Retardo entre símbolos
  }
}
void morse (char txtString[]) {
    for (int i = 0; txtString[i] != '\0'; i++) {
        if (txtString[i] >= 'a' && txtString[i] <= 'z')
        flashLED(Tabla[txtString[i] - 'a']);
      else if (txtString[i] >= 'A' && txtString[i] <= 'Z')
        flashLED(Tabla[txtString[i] - 'A']);
      else if (txtString[i] >= '0' && txtString[i] <= '9')
        flashLED(Tabla[txtString[i] - '0' + 26]);
      else if (txtString[i] == ',')
        flashLED(Tabla[36]);
      else if (txtString[i] == '.')
        flashLED(Tabla[37]);
      else if (txtString[i] == ' ')
        __delay_ms(60);                   //Retardo entre palabras. Se le suma al
                                          //del último símbolo
      __delay_ms(20);               //Retardo entre caracteres. Se le suma al del
                                    //último símbolo
    }
}
```

- *Comprueba el funcionamiento mediante la utilización del simulador MPLAB SIM y el visualizador de señales digitales (menú View⇒ Simulator Logic Analyzer) observando la salida RB0*

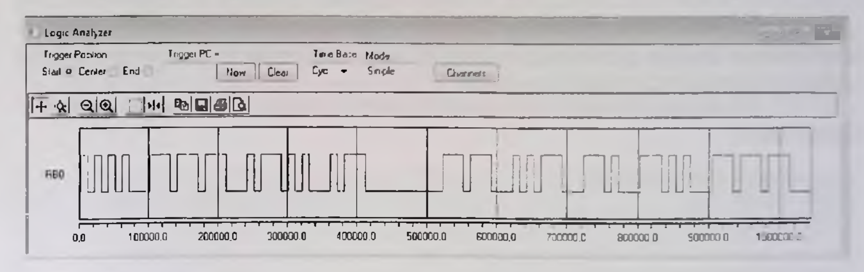


FIGURA 2.7: SIMULACIÓN DEL EJERCICIO 2.1.3.

2.1.4 Generación de la señal acústica SOS mediante un Zumbador Piezoeléctrico

Escribe un programa que genere la señal de socorro internacional SOS en código morse (. . . - - - . . .). La salida activará el zumbador piezoeléctrico conectado al puerto RC2, de modo indefinido hasta que se pulse el pulsador conectado a RB0 que generará una interrupción para finalizar el programa.

Los tiempos de cada pitido serán los siguientes:

- pulso corto: 200 ms
- pulso largo (raya): 500 ms
- espacio entre pulsos: 200 ms
- pausa entre dos señales SOS: 500 ms
- periodo de la onda cuadrada que excita el zumbador: 4 ms

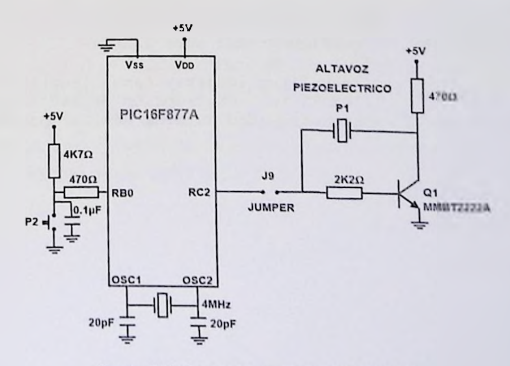


FIGURA 2.8: CONEXIONES CON EL PIC16F877A.

Solución:

Se generan tres funciones: una para el punto, otra para la raya y la última para la pausa entre las dos señales. En cada función se genera una onda cuadrada de semiperiodo 2 ms (__delay_ms(2))que activa el zumbador durante un tiempo controlado por un *for*. Mediante la interrupción externa producida por RB0/INT se deja de enviar la señal al zumbador al cambiar el terminal RC2 de salida a entrada.

Código fuente

```
//ARCHIVOS DE DEFINICIONES
#include<htc.h>                        //Incluimos librería del micro a usar
#define _XTAL_FREQ 4000000             //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

//DEFINICION DE VARIABLES
      int x;

//DEFINICION DE FUNCIONES
void punto(void){                      //Función para generar el punto
      for(x=0;x<100;x++){
         RC2 = ~RC2;
         __delay_ms(2);
      }
      __delay_ms(200);
}
void raya(void){                       //Función para generar la raya
      for(x=0;x<250;x++){
         RC2 = ~RC2;
         __delay_ms(2);
      }
      __delay_ms(200);
}
void pausa(void){                      //Función para generar la pausa entre dos SOS
         __delay_ms(500);
}

//PROGRAMA PRINCIPAL
void main(void){
      TRISB = 0b00000001;      //Configurar RB0 como entrada
      TRISC = 0b00000000;      /Configurar RC2 como salida
      RC2=1;
      GIE=1;                   //Habilitar interrupciones (Registro INTCON)
      INTE=1;                  //Habilitar interrupción de RB0 (Registro INTCON)
      INTEDG=0;                //Interrupción externa RB0/INT por flanco de bajada
      while(1){
         punto();
         punto();
         punto();
         raya();
         raya();
         raya();
         punto();
         punto();
         punto();
         pausa();
      }
}
```

```
//INTERRUPCIÓN
//Interrupcion por RB0 /INT
static void interrupt isr(void){
      if(INTF){
        INTF=0;                  //Desactivar el FLAG de la interrupción de RB0
        TRISC = 0b00000100;      //Configurar RC2 como entrada para parar
      }
}
```

Se puede observar el funcionamiento mediante el analizador lógico del simulador.

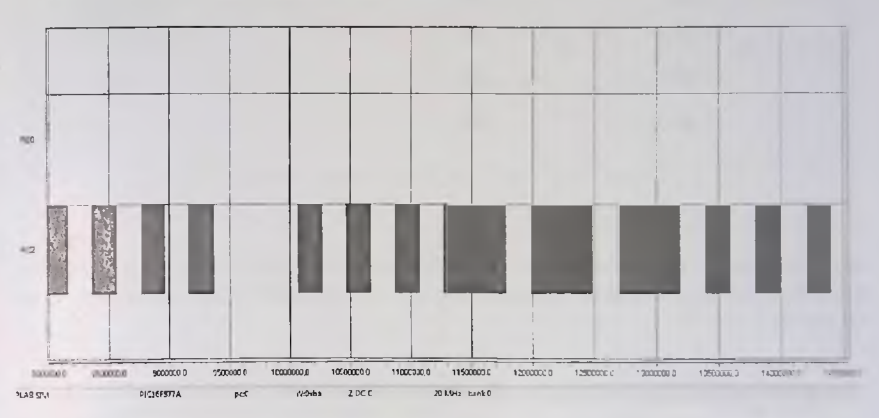


FIGURA 2.9: SIMULACIÓN DEL EJERCICIO 2.1.4.

2.2 Visualizadores numéricos de 7 segmentos

Los visualizadores de 7 segmentos con diodos LED se usan sobre todo para representar información numérica. Hay dos tipos de elementos de 7 segmentos: los de ánodo común y los de cátodo común, según estén conectados entre sí todos los ánodos o todos los cátodos de los LED, respectivamente. En los de ánodo común, para que se active (emita luz) un segmento, el terminal correspondiente debe excitarse con una tensión baja (correspondiente al nivel lógico '0', si la lógica es positiva), mientras que el ánodo común debe estar puesto a una tensión positiva alta (VDD). En los elementos de cátodo común la situación es la inversa: cada segmento se activa con una tensión alta (correspondiente al nivel lógico '1') y el cátodo común debe estar a 0 V (VSS).

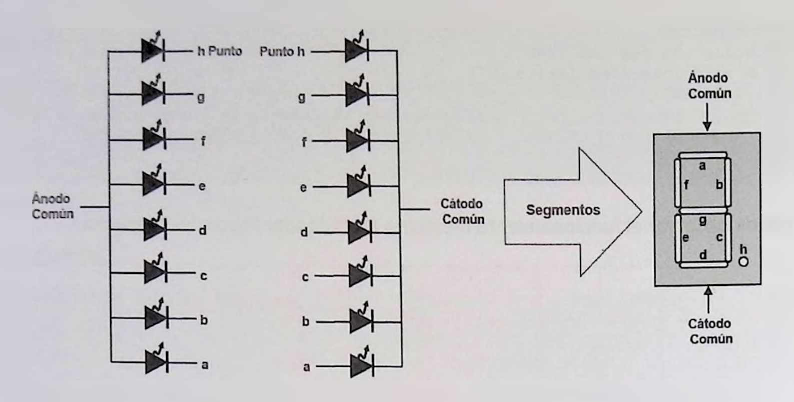
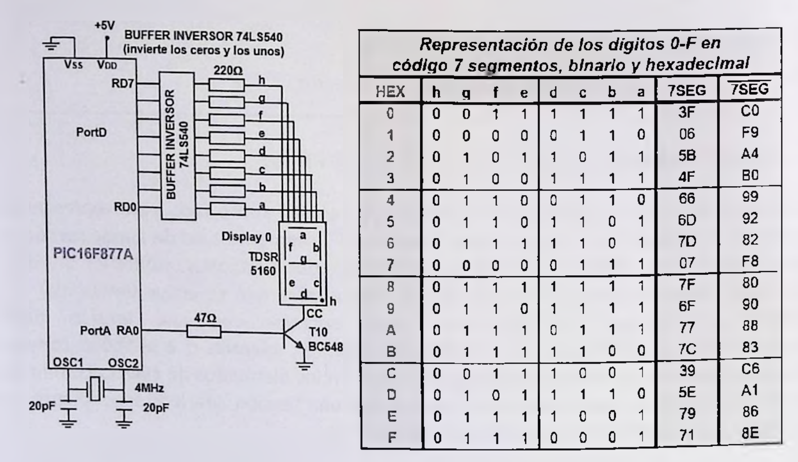


FIGURA 2.10: DIAGRAMA INTERNO DE UN DISPLAY 7 SEGMENTOS.

2.2.1 Control de 1 display

Muestra de manera indefinida por el *display* de 7 segmentos (figura 2.11) uno de los dígitos hexadecimales encendiéndose y apagándose con una cadencia de un segundo. El dígito elegido es E.



Representación de los digitos 0-F en código 7 segmentos, binario y hexadecimal										
HEX	h	g	f	e	d	c	b	a	7SEG	$\overline{7SEG}$
0	0	0	1	1	1	1	1	1	3F	C0
1	0	0	0	0	0	1	1	0	06	F9
2	0	1	0	1	1	0	1	1	5B	A4
3	0	1	0	0	1	1	1	1	4F	B0
4	0	1	1	0	0	1	1	0	66	99
5	0	1	1	0	1	1	0	1	6D	92
6	0	1	1	1	1	1	0	1	7D	82
7	0	0	0	0	0	1	1	1	07	F8
8	0	1	1	1	1	1	1	1	7F	80
9	0	1	1	0	1	1	1	1	6F	90
A	0	1	1	1	0	1	1	1	77	88
B	0	1	1	1	1	1	0	0	7C	83
C	0	0	1	1	1	0	0	1	39	C6
D	0	1	0	1	1	1	1	0	5E	A1
E	0	1	1	1	1	0	0	1	79	86
F	0	1	1	1	0	0	0	1	71	8E

FIGURA 2.11: CONEXIÓN DE UN DISPLAY DE 7 SEGMENTOS CON EL PIC16F877A. CÓDIGO 7 SEGMENTOS Y COMPLEMENTADO DE LAS CIFRAS HEXADECIMALES (0-F).

Solución:

Debido al inversor 74LS540 hay que enviar el complemento del código 7 segmentos de la letra E al PORTD.

Código fuente

```
//ARCHIVOS DE DEFINICIONES
#include<htc.h>                         //Incluimos librería del micro a usar
#include "ports.h"
#define _XTAL_FREQ 4000000              //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

//DEFINICIÓN DE VARIABLES
//Tabla con el código de 7 segmentos complementado para las cifras de 0 a F.
//(tamaño byte)
unsigned char codigo[]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90,
                                                0x88,0x83,0xC6,0xA1,0x86,0x8E};

unsigned char DIGITO;                   //Digito que se quiere sacar

//PROGRAMA PRINCIPAL
void main(void){
      set_all_digital();                //Desactiva PORTA como entradas analógicas
      TRISA=0b00000000;                 //Configura el PORTA como salidas
      TRISD=0x00;                       //Configura el PORTD como salidas
      PORTA=1;                          //Activar display de RA0
      DIGITO=0x0E;                      //Cargar E en DIGITO

      while(1){
      PORTD=codigo[DIGITO];             //Saca el DÍGITO por el puerto D
      __delay_ms(1000);                 //Dígito mostrado durante 1 segundo
      PORTD=0xFF;                       //Apagar el display
      __delay_ms(1000);                 //Display apagado durante 1 segundo
      }
}
```

Se puede observar el funcionamiento mediante el analizador lógico del simulador.

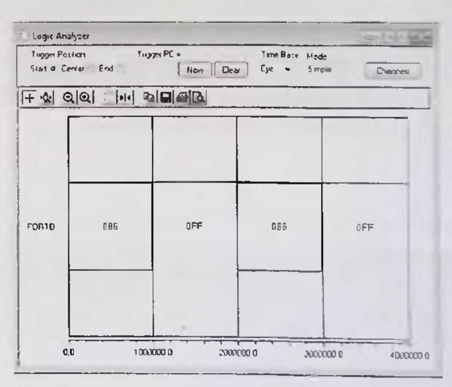


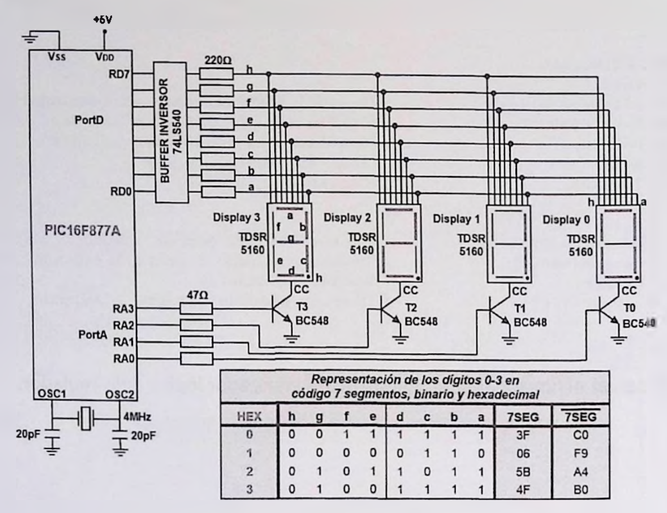
FIGURA 2.12: SIMULACIÓN DEL EJERCICIO 2.2.1.

2.2.2 Control de un visualizador con 4 displays

La figura 2.13 muestra un visualizador o display de 4 elementos de cátodo común conectados a los puertos A y D de un microcontrolador PIC. Las resistencias Rc = 220 Ω

limitan la corriente que circula por los segmentos, mientras que las resistencias Rb = 47 Ω deben garantizar la saturación de los transistores que activan los terminales de selección de cada elemento. Los transistores actúan como interruptores y van a saturación con un nivel lógico ‘1’ en los terminales del puerto A y a corte con un nivel lógico ‘0’. La frecuencia mínima de las señales en cada uno de terminales de selección (RA3, RA2, RA1 y RA0) debe estar entre 40 Hz y 200 Hz para que no se perciba parpadeo alguno. Cada elemento está seleccionado durante una cuarta parte del tiempo. Si consideramos una frecuencia de 50 Hz, cada uno de los displays se refrescará cada 20 ms y estará seleccionado durante 5 ms.

Se pide mostrar en el visualizador los dígitos 3-2-1-0.



Representación de los dígitos 0-3 en código 7 segmentos, binario y hexadecimal										
HEX	h	g	f	e	d	c	b	a	7SEG	$\overline{7SEG}$
0	0	0	1	1	1	1	1	1	3F	C0
1	0	0	0	0	0	1	1	0	06	F9
2	0	1	0	1	1	0	1	1	5B	A4
3	0	1	0	0	1	1	1	1	4F	B0

FIGURA 2.13: MULTIPLEXACIÓN DE DISPLAYS.

Soluciones:

Código fuente 1

```
//ARCHIVOS DE DEFINICIONES
#include<htc.h>                         //Incluimos librería del micro a usar
#include "ports.h"
#define _XTAL_FREQ 4000000                   //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);   //Configuración
                                                               //del PIC

//PROGRAMA PRINCIPAL
void main(){
      //INICIALIZACIONES PARA EL PIC
      set_all_digital();                     //Desactivamos PORTA como entradas
                                             //analógicas
      TRISA=0x00;                            //Terminales PORTA como salidas
      TRISD=0x00;                            //Terminales PORTD como salidas
```

```
        while(1){
           //Código para mostrar los dígitos y activar los displays
           RA3=0;RA2=0;RA1=0;RA0=1;  //Activamos DISPLAY 0 y los otros desactivados
           PORTD = 0xC0;             //Mostramos por puertoD el dígito 0
           __delay_ms(5);            //Retardo de 5 ms para evitar el parpadeo

           RA3=0;RA2=0;RA1=1;RA0=0;  //Activamos DISPLAY 1 y los otros desactivados
           PORTD = 0xF9;             //Mostramos por puertoD el dígito 1
           __delay_ms(5);            //Retardo de 5 ms para evitar el parpadeo

           RA3=0;RA2=1;RA1=0;RA0=0;  //Activamos DISPLAY 2 y los otros desactivados
           PORTD = 0xA4;             //Mostramos por puertoD el dígito 2
           __delay_ms(5);            //Retardo de 5 ms para evitar el parpadeo

           RA3=1;RA2=0;RA1=0;RA0=0;  //Activamos DISPLAY 3 y los otros desactivados
           PORTD = 0xB0;             //Mostramos por puertoD el dígito 3
           __delay_ms(5);            //Retardo de 5 ms para evitar el parpadeo
        }
}
```

Se puede observar el funcionamiento mediante el analizador lógico del simulador.

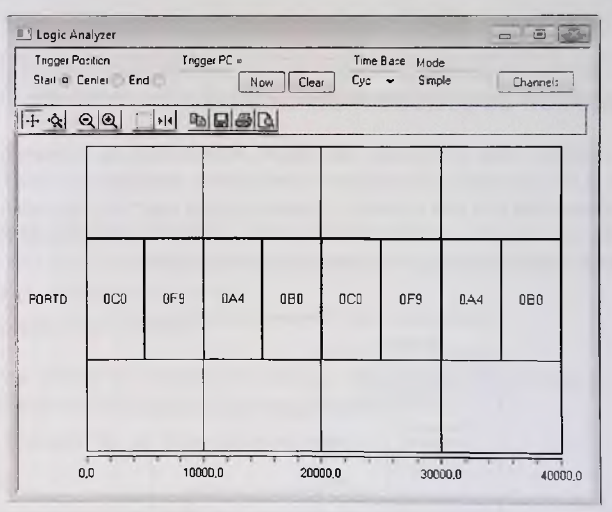


FIGURA 2.14: SIMULACIÓN DEL EJERCICIO 2.2.2.

Código fuente 2

```
//ARCHIVOS DE DEFINICIONES
#include<htc.h>                          //Incluimos librería del micro a usar
#include "ports.h"
#define _XTAL_FREQ 4000000               //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);   //Configuración
                                                                //del PIC
```

```
//Tabla con el código 7segmentos complementado
unsigned char codigo[]={0xC0,0xF9,0xA4,0xB0};      //0, 1, 2 y 3
//Tabla con el código de activación de los display
unsigned char display[]={0x01,0x02,0x04,0x08};     //Display 0, display 1,
                                                   //display 2 y display 3

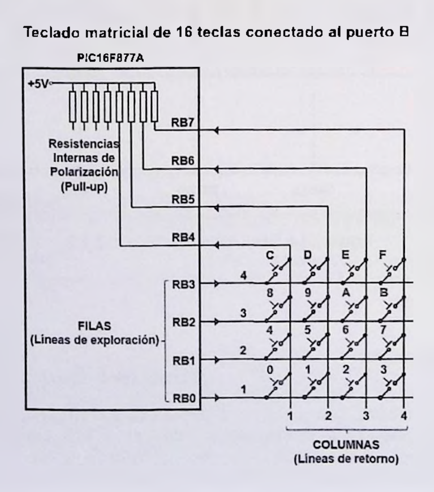
unsigned char x=0;                        //Índice de los códigos

// PROGRAMA PRINCIPAL
void main(){
      //INICIALIZACIONES PARA EL PIC
      set_all_digital();                  //Desactivamos PORTA como entradas
                                          //analógicas
      TRISA=0x00;                         //Terminales PORTA como salidas
      TRISD=0x00;                         //Terminales PORTD como salidas

      while(1){
        for(x=0;x<4;x++){
        PORTA=display[x];                 //Activar display
        PORTD=codigo[x];                  //enviar código
        __delay_ms(5);                    //Display y código activado 5 ms
        }
      }
}
```

2.3. Exploración de teclado matricial

Un teclado matricial está compuesto por teclas interconectadas formando una matriz (véase figura 2.15). Las teclas son simples interruptores mecánicos y cada una ocupa la intersección de una fila con una columna. Cuando se pulsa una tecla, se ponen en contacto eléctrico la fila y la columna donde está dicha tecla. Las filas y columnas de esta matriz se pueden conectar a los terminales de uno o más puertos paralelos.



LECTURA TECLADO			
Si RB0 = 0 y	tecla '0' pulsada	⇒	RB4 = 0
	tecla '0' no pulsada	⇒	RB4 = 1
Si RB0 = 1 y	tecla '0' pulsada	⇒	RB4 = 1
	tecla '0' no pulsada	⇒	RB4 = 1

FIGURA 2.15: EXPLORACIÓN DE TECLADO MATRICIAL CONECTADO AL PUERTO B.

Para explorar un teclado matricial se envían señales hacia las filas de la matriz por las líneas de exploración y se recoge información por las columnas, que entonces constituyen las líneas de retorno. Básicamente se parte de que, si no hay ninguna tecla pulsada, todas las líneas de retorno están en el nivel lógico '1'. Las líneas de exploración son puestas (sucesiva o simultáneamente) a '0'. Este valor lógico solo aparece en la línea de retorno donde está la tecla pulsada, mientras que las restantes líneas de retorno mantienen el valor '1'. Con la información enviada hacia la matriz y la que retorna, se conforma un código único para cada tecla, llamado código de exploración. Para garantizar que las líneas de retorno permanezcan en '1' si no hay tecla pulsada, se conectan resistencias entre cada línea de retorno y la tensión de alimentación (VDD).

El código que retornará el teclado será el valor de la tecla pulsada. Conectando al puerto D un display de 7 segmentos (véase figura del ejemplo 2.2.1) se podrá ver dicho valor en el display.

Configuración

Hay que configurar los terminales de los puertos empleados:

B: RB7-RB4 como terminales de entrada y RB3-RB= como terminales de salida,

D: todos los terminales de salida y

A: el terminal RA0 de salida.

Como se van a utilizar las resistencias internas de polarización del puerto B, hay que indicarlo en el registro OPTION_REG (véase apéndice A2.2)

Con OPTION_REG=0x7F se habilitan dichas resistencias.

Código fuente

```
//ARCHIVOS DE DEFINICIONES
#include<htc.h>                     //Incluimos librería del micro a usar
#include "ports.h"
#define _XTAL_FREQ 4000000          //Oscilador Interno de 4MHZ

__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

//DEFINICIÓN DE VARIABLES
//Tabla con el código de 7 segmentos complementado para los dígitos
//del 0 al F y apagado.
unsigned char codigo[]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,
                                        0x98,0x88,0x83,0xC6,0xA1,0x86,0x8E,0xFF};
unsigned char tecla;                //Índice de lista de código
```

```
//PROGRAMA PRINCIPAL
void main(void){
      set_all_digital();            //Hay que configurar PA0-3 como digitales
      TRISA = 0b00000000;           //Configura el PORTA como salidas
      TRISD=0x00;                   //Configura el PORTD como salidas
      TRISB=0xF0;                   //Configura como salidas RB0-RB3 y como
                                    //entradas RB4-RB7
      OPTION_REG=0x7F;              //Habilita RESIS. De pull-up internas del
                                    //puerto B.
      PORTA=1;                      //Activar display de RA0

      while(1){                     //Bucle infinito.

      PORTB=0xFE; // Saca 0 a Fila 1
      if(RB4 == 0){tecla = 0;}
      if(RB5 == 0){tecla =1;}
      if(RB6 == 0){tecla = 2;}
      if(RB7 == 0){tecla = 3;}
      PORTB=0xFD; // Saca 0 a Fila 2
      if(RB4 == 0){tecla = 4;}
      if(RB5 == 0){tecla = 5;}
      if(RB6 == 0){tecla = 6;}
      if(RB7 == 0){tecla = 7;}
      PORTB=0xFB; // Saca 0 a Fila 3
      if(RB4 == 0){tecla = 8;}
      if(RB5 == 0){tecla = 9;}
      if(RB6 == 0){tecla = 10;}
      if(RB7 == 0){tecla = 11;}
      PORTB=0xF7; // Saca 0 a Fila 4
      if(RB4 == 0){tecla = 12;}
      if(RB5 == 0){tecla = 13;}
      if(RB6 == 0){tecla = 14;}
      if(RB7 == 0){tecla = 15;}
      PORTD = codigo[tecla];        //Sacar código de 7 segmentos al display
      if((RB4 && RB5 && RB6 && RB7)== 1){tecla = 16;}      //Apagado el display

      }
}
```

2.4 Control de pantallas LCD

Para los siguientes ejercicios nos vamos a basar en la conexión con un display LCD de dos líneas y 16 caracteres por línea el cual está diseñado sobre el controlador Hitachi HD44780.

Consultar las hojas de características en:

https://www.sparkfun.com/datasheets/LCD/HD44780.pdf

http://en.wikipedia.org/wiki/Hitachi_HD44780_LCD_controller

El display dispone de 14 líneas para su alimentación, control del contraste y para la comunicación con el microprocesador. Cada carácter consiste en una matriz de puntos de 5 × 8.



FIGURA 2.16: PANTALLA LCD.

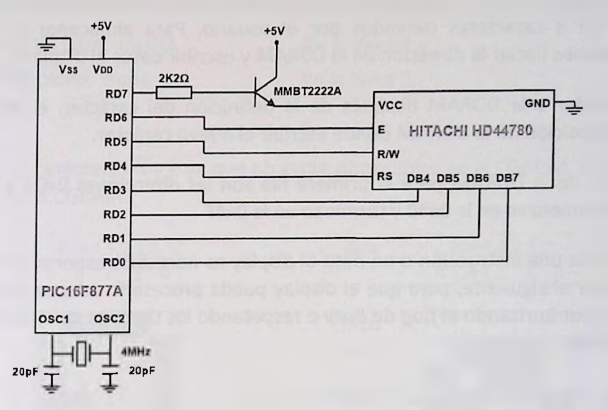


FIGURA 2.17: CONEXIÓN DEL PIC16F877A CON LA PANTALLA LCD HITACHI HD 44780.

La comunicación con el microprocesador se ha implementado empleando un bus de datos de 4 bits.

En la conexión con el microprocesador el puerto D controla el display mediante las siguientes líneas:

- Encendido / Apagado del display: RD7
- Bus de datos: RD0 – RD3
- Bus de control:
  - Enable (E): RD6: Cuando está a nivel bajo el display está deshabilitado.
  - Read/Write (R/W): RD5: cuando está a nivel bajo los datos se escriben en el LCD. Cuando es alto, los datos se leen del LCD.
  - Register Select (RS): RD4: Sirve para distinguir entre instrucciones y caracteres. A nivel bajo se envían instrucciones y a nivel alto caracteres.

Cuando el LCD se inicializa está preparado para recibir caracteres. Si recibe un carácter lo escribe en el display y mueve el cursor a la siguiente posición. Los caracteres mostrados en el display se almacenan en la memoria DDRAM.

El display LCD contiene tres bloques de memoria:

- DDRAM – Display Data RAM
- CGRAM – Character Generator RAM
- CGROM – Character Generator ROM

Para mandar un carácter al display, se escribe en la DDRAM. La CGROM contiene las matrices de puntos del juego de caracteres por defecto.

También es posible generar nuevos caracteres definidos por el usuario mediante la CGRAM, una memoria de 64 bytes. Cada carácter necesita 8 bytes para almacenarse, por lo que disponemos de 8 caracteres definidos por el usuario. Para almacenar el mapa de un carácter, debemos iniciar la dirección de la CGRAM y escribir datos al display.

Antes de acceder a la DDRAM después de la definición del carácter, el programa debe establecer la posición de la DDRAM donde escribir el nuevo carácter.

Las posiciones de la DDRAM para la primera fila son las direcciones 0x00 a 0x0F. Para la segunda fila comienzan en la 0x40 y terminan en la 0x4F

Cuando se envía una instrucción o un dato al display es necesario esperar un cierto tiempo antes de enviar el siguiente, para que el display pueda procesarlo adecuadamente. Esto se puede hacer monitorizando el *flag* de *Busy* o respetando los tiempos de ejecución indicados por el fabricante.

2.4.1 Envío de cadenas de caracteres al LCD

Realiza un programa que escriba en las dos líneas de una pantalla LCD 16x2.

Solución:

En la línea 1 aparecerá el texto "LA PRIMERA LÍNEA" y en la línea 2 el texto "La segunda línea".

Al crear el proyecto hay que cargar además del fichero con el programa los ficheros lcd.c y lcd.h

Código fuente

```
//ARCHIVOS DE DEFINICIONES
#include <htc.h>                    //Incluimos librería del micro a usar.
#include "lcd.h"                    //Incluimos librería de la pantalla lcd.
#define _XTAL_FREQ 4000000          //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

//PROGRAMA PRINCIPAL
void main (void){

      lcd_init();                   //Inicializa la pantalla LCD.
      lcd_clear();                  //Borra el contenido de la pantalla lcd.
```

```
    while(1){
      lcd_goto(0x00);            //Escribe en la primera línea de la pantalla
                                 //LCD (de 0x00 a 0x0F)
      lcd_puts("LA PRIMERA LINEA");   //Presentación de la pantalla.
      lcd_goto(0x40);                 //Escribe en la segunda línea de la
                                      //pantalla LCD (de 0x40 a 0x4F)
      lcd_puts("La segunda linea ");  //Presentación de la pantalla.
    }
}
```

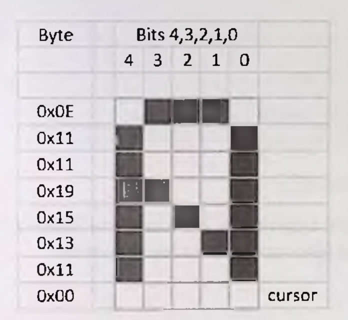
2.4.2 Creación de nuevos caracteres

Realiza un programa que escriba en las dos líneas de la pantalla LCD los mensajes

PAMPLONA IRUÑA	En la línea 1 y
Pamplona Iruña	En la línea 2.

Solución:

Hay que crear los caracteres Ñ y ñ ya que no están disponibles en la CGROM. Para ello los vamos a crear en la CGRAM.



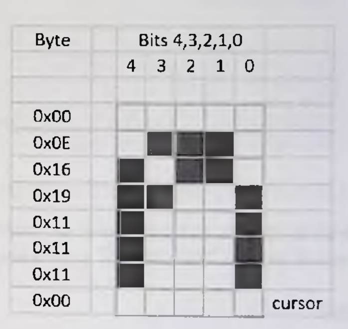


FIGURA 2.18: CARACTERES Ñ Y ñ.

Código fuente

```
//ARCHIVOS DE DEFINICIONES
#include <htc.h>
#include "lcd.h"
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

#define _XTAL_FREQ 4000000        //Oscilador Interno de 4MHZ
#define LCD_RS RD4

//Definición de caracteres en arrays de 8 bytes (5x8)
//sólo se tienen en cuenta los últimos 5 bits.
unsigned char MAY[8] = {0x0E,0x11,0x11,0x19,0x15,0x13,0x11,0x00};
unsigned char MIN[8] = {0x00,0x0E,0x16,0x19,0x11,0x11,0x11,0x00};

//DEFINICION DE FUNCIONES
//LcdDefineChar: Función para cargar el nuevo carácter en CGRAM
//PARAMETRO1: Posición del carácter, 0-7.
```

```
//PARAMETRO2: Carácter bitmap (8 bytes). (5x8)
LcdDefineChar(unsigned char charnum, char values[]){
      int i;
//selecciona posición CGRAM
      LCD_RS=0;
      lcd_write(0x40+8*charnum);         //dispCmd(0x40 + charnum*8);
//introduce el carácter en CGRAM
      for (i=0; i<8; i++) {
        lcd_putch(values[i]);
      }
}
//PROGRAMA PRINCIPAL
void main(void){
      lcd_init();                  //Inicializa la pantalla LCD.
      lcd_clear();                 //Borra el contenido de la pantalla LCD
      LcdDefineChar(0,MAY);        //Introduce el nuevo carácter Ñ mayúscula
                                   //en posición 0 y nombre MAY
      LcdDefineChar(1,MIN);        //Introduce el nuevo carácter ñ minúscula
                                   //en posición 1 y nombre MIN

      lcd_goto(0x00);              //Selecciona la primera línea para escribir
      lcd_puts("PAMPLONA   IRU");
      lcd_putch(0x00);             //Escribe la Ñ
      lcd_puts("A");

      lcd_goto(0x40);              //Selecciona la segunda línea para escribir
      lcd_puts("Pamplona   Iru");
      lcd_putch(0x01);             //Escribe la ñ
      lcd_puts("a");
      while(1);                    //Bucle infinito
}
```

2.4.3 Cronómetro segundero en la pantalla LCD

Realiza un programa que para crear un cronómetro segundero. En la primera línea de la LCD aparecerá el letrero "SEGUNDOS" y en la segunda línea aparecerán los segundos.

Solución:

Al crear el proyecto hay que cargar además del fichero.c con el programa los ficheros lcd.c y lcd.h

Código fuente

```
//ARCHIVOS DE DEFINICIONES
#include <htc.h>                 //Incluimos librería del micro a usar.
#include "lcd.h"                 //Incluimos librería de la pantalla lcd.
#define _XTAL_FREQ 4000000       //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

//DEFINICION DE VARIABLES
unsigned char x,segundos,decenas,unidades;     //Variables que usaremos

//PROGRAMA PRINCIPAL
void main (void){

      lcd_init();                //Inicializa la pantalla LCD.
      lcd_clear();               //Borra el contenido de la pantalla lcd.
```

```
      lcd_goto(0x04);              //Escribe en la pantalla LCD la presentación
                                   //inicial.
      lcd_puts("SEGUNDOS");        //Presentación de la pantalla.
      while(1){
         for(x=0;x<60;x++){
                segundos=x;              //Activar display
//Código que nos sirve para pasar la variable segundos a BDC
                decenas=segundos/10;     //Guardamos las decenas de segundo
                segundos=segundos%10;
                unidades=segundos;       //Guardamos las unidades de segundo
//Sumamos 0x30 para obtener su código ascii.
                decenas=decenas+0x30;
                unidades=unidades+0x30;
//Código para mostrar los valores por la pantalla lcd.
                lcd_goto(0x47);
                lcd_putch(decenas);
                lcd_goto(0x48);
                lcd_putch(unidades);
                __delay_ms(1000);        //Retardo de 1 segundo
         }
      }
}
```

3. Temporizadores

Los temporizadores del PIC, junto a su sistema de interrupciones, ayudan a construir sistemas que realizan varias tareas dependientes del tiempo simultáneamente. Un ejemplo es un cronómetro que usa displays de 7-segmentos para visualizar la cuenta. Por un lado, cuenta el tiempo de forma muy precisa y por otro refresca los displays de manera periódica. Para realizar estas tareas el microcontrolador debe usar un temporizador que le indique el momento exacto en que debe iniciar cada rutina.

El PIC 16F877A tiene tres temporizadores, Timer0, Timer1 y Timer2, que se pueden utilizar para crear retardos, realizar tareas periódicas, generar señales PWM o contar pulsos externos. Al tratarse de dispositivos *hardware* independientes de la CPU, todas estas funciones tienen lugar de manera simultánea, y a la vez que el micro puede estar realizando otra tarea.

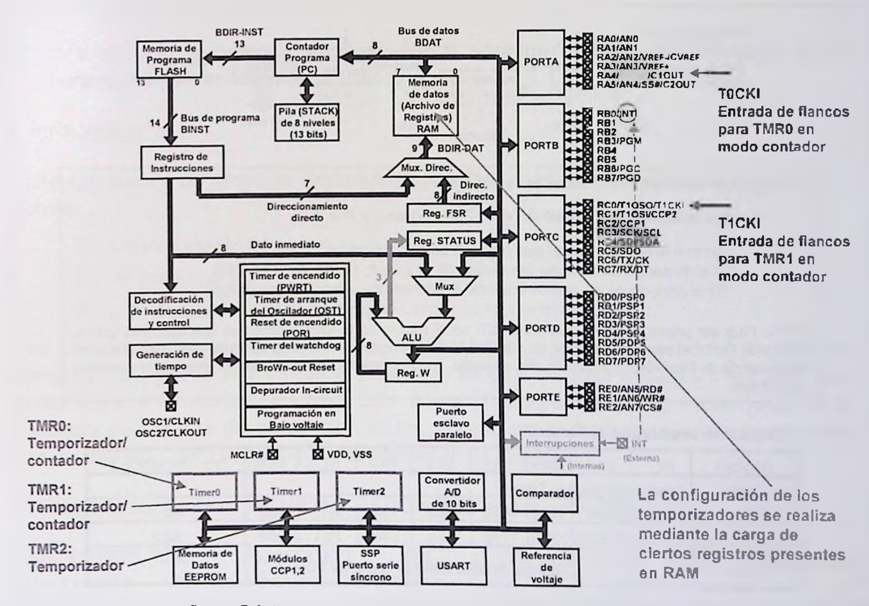


FIGURA 3.1: LOS TEMPORIZADORES DEL MICROCONTROLADOR PIC16F877A.

3.1. Temporizador 0

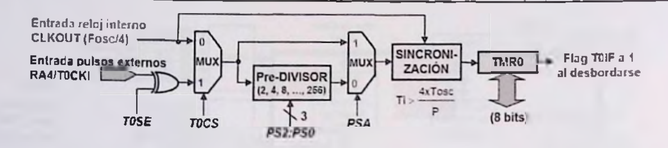
El Temporizador 0 dispone de un contador/temporizador de 8 bits además de un predivisor de 8 bits programable. La fuente de pulsos puede ser interna o externa. En este segundo caso, se sincroniza con el reloj interno y es posible seleccionar el flanco en que se produce la cuenta.

El funcionamiento como temporizador se selecciona poniendo a cero el bit T0CS del registro OPTION. En este modo el temporizador se incrementa en cada ciclo de instrucción (sin tener en cuenta el pre-divisor). Cuando se escribe en el registro TMR0, la cuenta se inhibe durante los dos siguientes ciclos.

El modo contador se selecciona poniendo a "1" el bit T0CS. En este modo cuenta los pulsos que se aplican al pin T0CKI. El flanco se determina mediante el bit T0SE del registro OPTION.

Para asignar el pre-divisor al Temporizador 0 es necesario borrar el bit PSA del registro OPTION. En otro caso se asigna al Watchdog.

Cuando la cuenta se desborda (pasa de 0xFF a 0x00) se pone a 1 el flag T0IF del registro INTCON. T0IF debe ser borrado por la rutina de atención a la interrupción para rehabilitar esta interrupción.



El tiempo que tarda en desbordarse el Timer0 es:

Temporización del Timer0 = Td = (256 - N_{TMR0}) x P x T

N_{TMR0} el número con el que hay que cargar el registro TMR0.
P el factor de división del pre-divisor *(P = 2, 4, 8, 16, 32, 64, 128, 256)*.
Ti el período de los pulsos internos o externos de entrada al pre-divisor.

NOTA: Para ser precisos, habría que añadir 2Ti en el cálculo de la temporización ya que cada vez que se recarga el TMR0 se pierden dos ciclos de máquina hasta el siguiente incremento como se explica en el manual de referencia de Microchip® (DS33023). Sin embargo, en los ejercicios propuestos se considerará despreciable este tiempo.

Registros asociados al Timer0

Registro	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Ver APENDICE 2
TMR0	Registro del módulo Timer0								
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	A2.5
OPTION_REG	RBPU#	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	A2.2
TRISA			Registro de direcciones de datos del PORTA						

Figura 3.2: Representación en bloques del funcionamiento del Timer 0.

3.1.1 Utilización del Temporizador 0 sin interrupción

Se pretende activar y desactivar el LED D1 del circuito de la figura correspondiente a la placa de PICDEM2+ con una cadencia de 1 segundo.

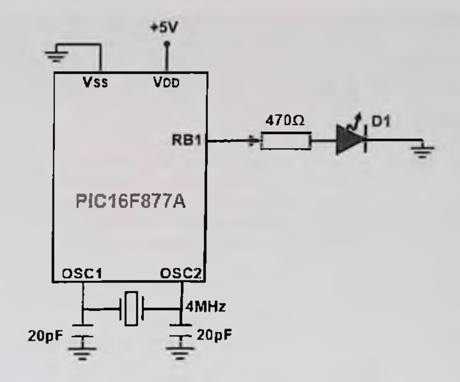


FIGURA 3.3: CONEXIÓN DEL LED AL PIC16F877A PARA ENCENDERLO Y APAGARLO CON EL TIMER0.

Solución:

En este ejercicio la cadencia de encendido y apagado del LED se conseguirá con el temporizador 0 y sin utilizar interrupciones.

Configuración

La temporización o tiempo de desbordamiento del Timer0 es Td = (256 - N_{TMR0}) * P * Ti donde

Ti es el período de los pulsos,

P es el predivisor,

N_{TMR0} es el valor con el que hay que cargar el temporizador 0.

Con un oscilador de Fosc = 4 MHz (Ti = 4/4 MHz = 1 microsegundo), la máxima temporización que se alcanza con el Timer0 es de 65,5 ms (con N_{TMR0} = 0 y P = 256). Para poder llegar a temporizar 1 s hace falta que el Timer0 se desborde varias veces. Si se configura para que se desborde cada 50 ms, al cabo de 20 desbordamientos (controlados con un contador cont) se tendrá la temporización de 1 segundo.

El Timer0 se configura con el registro OPTION_REG (véase apéndice A2.2)

Con OPTION_REG = 0b11000111 = 0xC7 el Timer0 actúa como temporizador (Ti = 4/4 MHz = 1 μs) y con predivisor P = 256.

Para conseguir que el tiempo de desbordamiento sea de 50 ms hay que cargar TMR0 con $N_{TMR0} = 60$.

Código fuente

```
//ARCHIVOS DE DEFINICIONES
#include<htc.h>                      //Incluimos librería del micro a usar
#define _XTAL_FREQ 4000000           //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

//PROGRAMA PRINCIPAL
void main(void){
      unsigned char cont;            //Contador de desbordamiento de TMR0
      TRISB=0x00;                    //Configura el PORTB como salidas
      PORTB=0;                       //Todos los led apagados
      OPTION_REG=0xC7;               //TMR0 como temporizador con predivisor P=256
      TMR0=60;                       //Para que Td sean 50ms
      while(1){
         if(T0IF){
                 T0IF=0;             //Desactivar el FLAG de TMR0
                 TMR0=60;            //Recargar TMR0
                 cont++;             //Incrementar contador de desbordamiento de TMR0
                 if(cont==20){       //Si se han llegado a 20 desbordamiento
                                     //(1 segundo)
                 PORTBbits.RB1 = ~PORTBbits.RB1;     //Cambia estado de RB1
                 cont=0;             // Reinicia contador de desbordamientos
                 }
         }
      }
}
```

Se puede observar el funcionamiento mediante el analizador lógico del simulador.

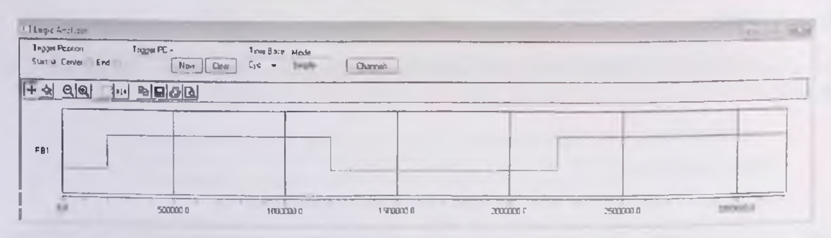


FIGURA 3.4: SIMULACIÓN DEL EJERCICIO 3.1.1.

3.1.2 Utilización del Temporizador 0 con interrupción

Muestra en el display de 7 segmentos (véase figura del ejercicio 2.2.1) los dígitos del 9 al 0 de manera indefinida. Los dígitos cambiarán con una cadencia de 2 segundos.

Solución:

La cadencia de cambio de los dígitos se conseguirá con el temporizador 0 y, en este caso, se utilizarán interrupciones.

Configuración

Con OPTION_REG = 0b11000111 = 0xC7 el Timer0 actúa como temporizador (Ti = 4/4 MHz = 1 μs) y con predivisor P = 256.

Para conseguir que el tiempo de desbordamiento (interrupción) sea de 50 ms hay que cargar TMR0 con N_{TMR0} = 60.

Para conseguir la cadencia de 2 segundos será necesario un contador *cont* que cuente 40 interrupciones.

Código fuente

```
//ARCHIVOS DE DEFINICIONES
#include<htc.h>                     //Incluimos librería del micro a usar
#include "timers.h"                 //Incluimos librería de timers
#define _XTAL_FREQ 4000000          //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

//DEFINICIÓN DE VARIABLES
//Tabla con el código de 7 segmentos complementado para los dígitos del 0 al 9.
//(tamaño byte)
unsigned char codigo[]={0b11000000,0b11111001,0b10100100,0b10110000,0b10011001,
0b10010010,0b10000011,0b11111000,0b10000000,0b10011000};
unsigned char x;                    //Índice de lista de código
unsigned char cont;                 //Contador de desbordamiento de TMR0

//PROGRAMA PRINCIPAL
void main(void){
      ADCON1=6;                     //Desactiva PORTA como entradas analógicas
      TRISA = 0b00000000;           //Configura el PORTA como salidas
      TRISD=0x00;                   //Configura el PORTD como salidas
      PORTA=1;                      //Activar display de RA0
      setup_timer0(RTCC_INTERNAL|RTCC_DIV256);      // TMR0 como temporizador
                                                    //predivisor=256
      set_timer0(60);               //Para que Td sean 50ms
      ei();                         //Habilitar interrupciones
      T0IE=1;                       //Habilitar interrupción de TMR0
      x=9;                          //Inicio índice de la lista con 9.
      PORTD=codigo[x];              //Saca el valor que corresponde por el puerto D
      while(1);                     //Bucle infinito.
}
```

```
//INTERRUPCIÓN por Timer0
static void interrupt isr(void){
      if(T0IF){
        T0IF=0;              //Desactivar el FLAG de la interrupción de TMR0
        set_timer0(60);      //Recargar TMR0
        cont++;              //Incrementar contador de interrupciones
        if(cont==40){
              x--;                 //Decremento índice lista
              PORTD=codigo[x];     //Saca el valor que corresponde por el puerto D
              if(x==0){
               x=10;               //Siguiente al último de la lista
              }
              cont=0;              //Ponemos a cero el contador de interrupciones
        }
      }
}
```

Se puede observar el funcionamiento mediante el analizador lógico del simulador.

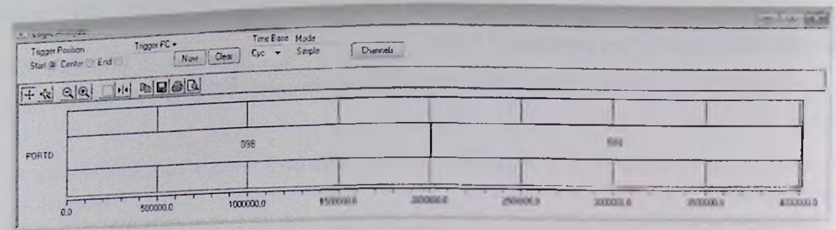


FIGURA 3.5: SIMULACIÓN DEL EJERCICIO 3.1.2.

3.1.3 Utilización del Temporizador 0 como contador de pulsos

Realiza un contador ascendente de dos dígitos que se incremente cada vez que se accione el pulsador conectado al puerto A (RA4) (figura 3.4). El valor del contador queda reflejado en el encendido de los displays.

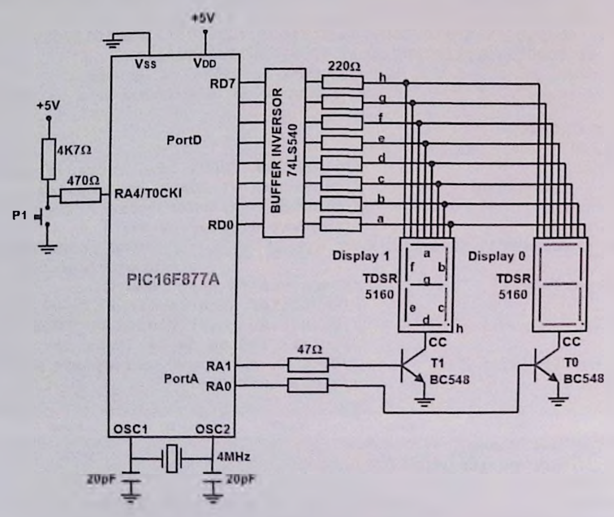


FIGURA 3.6: CONEXIONES CON EL PIC16F877A DEL PULSADOR QUE INTRODUCE LOS PULSOS EXTERNOS AL TIMER 0 Y DE LOS DISPLAYS DE 7 SEGMENTOS QUE MUESTRAN LOS PULSOS INTRODUCIDOS.

Solución:

En este ejercicio se utiliza el temporizador 0 como contador de los pulsos que entran por RA4/TOCKI.

Con cada pulso se desborda el Timer0 y se produce la interrupción que permite incrementar la cuenta de los displays.

Configuración

El Timer0 se configura con el registro OPTION_REG (véase apéndice A2.2)

Con OPTION_REG = 0b11111000 = 0xF8 el Timer0 actúa como contador de los pulsos externos que entran por T0CKI. El pre-divisor vale P = 1 ya que se asigna al Watchdog. Cargando TMR0 con N_{TMR0} = 255, el desbordamiento se producirá con cada pulso externo.

Código fuente

```
//ARCHIVOS DE DEFINICIONES
#include<htc.h>                       //Incluimos librería del micro a usar
#include "timers.h"                   //Incluimos librería de timers
#include "ports.h"                    //Incluimos librería de puertos
#define _XTAL_FREQ 4000000            //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

//DEFINICIÓN DE VARIABLES
//Tabla con el código de 7 segmentos complementado /para los dígitos del 0 al 9.
//(tamaño byte)

unsigned char codigo[]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
unsigned char cont = 0;       //Contador de interrupciones
unsigned char C2=0,C1=0;      //Llevamos la cuenta de 00 a 99

//PROGRAMA PRINCIPAL
void main(void){
      set_all_digital();      //Desactiva PORTA como entradas analógicas
      TRISA=0x10;             //PORTA con RA4 como entrada y resto como salidas
      TRISD=0x00;             //Terminales PORTD como salidas

//Configurar el TIMER0 como contador de pulsos externos por RA4 con predivisor 1
      setup_timer0(RTCC_EXT_FALL|RTCC_DIV1);
      set_timer0(255);

      ei();                   //Habilitar interrupciones
      T0IE=1;                 //Habilitar interrupción de TMR0

      while(1){               //Bucle infinito
//Código para mostrar los dígitos
         RA1=0;RA0=1;               //Activamos DISPLAY 0 y los otros desactivados
         PORTD = codigo[C1];        //Mostramos por puertoD las unidades
         __delay_ms(5);             //Retardo de 5 ms para evitar el parpadeo

         RA1=1;RA0=0;               //Activamos DISPLAY 1 y los otros desactivados
         PORTD = codigo[C2];        //Mostramos por puertoD las decenas
         __delay_ms(5);             //Retardo de 5 ms para evitar el parpadeo
      }
}
//INTERRUPCIÓN por Timer0
static void interrupt isr(){
      if(T0IF){
         T0IF=0;              //Desactivamos el FLAG de la interrupción de TMR0
         set_timer0(255);     //Recargamos TMR0
         C1++;                //incremento contador pulsos
         if(C1==10){
                 C1=0;              //Reseteamos las unidades
                 C2++;              //Incrementamos las decenas
```

```
                if(C2==10){          //Si hemos llegado a 10 decenas
                 C2 = 0;             //Reseteamos las decenas
                }
         }
      }
}
```

Se puede observar el funcionamiento mediante el analizador lógico del simulador donde se ha configurado RA4 como entrada de estímulo externo en modo de pulso alto de duración 100 ciclos de instrucción.

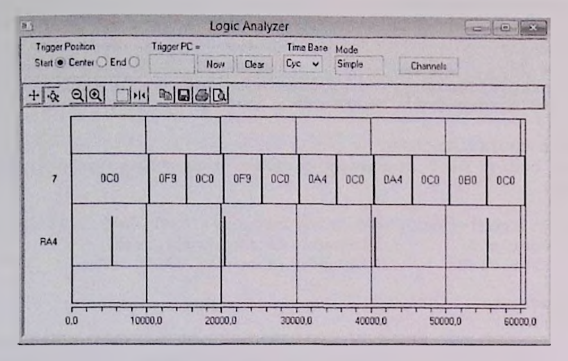


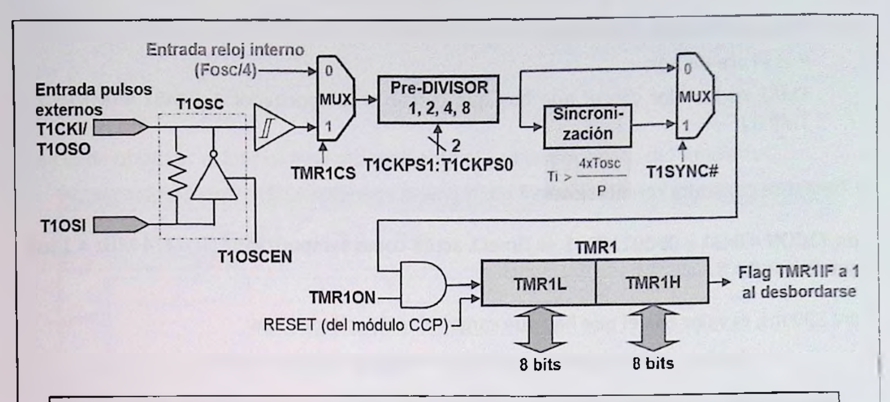
FIGURA 3.7: SIMULACIÓN DEL EJERCICIO 3.1.3.

3.2. Temporizador 1

El Temporizador 1 es un módulo que contiene un temporizador/contador de 16 bits accesible mediante dos registros de 8 bits (TMR1H:TMR1L). Cuando se desborda (pasando de 0xFFFF a 0x0000) se activa el flag de interrupción (TMR1IF). Esta interrupción se puede habilitar o deshabilitar mediante el bit TMR1IE.

Dispone de tres modos de funcionamiento:

- Temporizador síncrono.
- Contador síncrono
- Contador asíncrono



El tiempo que tarda en desbordarse el Timer1 es:

Temporización del Timer1 = Td = (65536 – N_{TMR1}) * P * T

N_{TMR1} es el valor que hay que cargar en TMR1 en su conjunto al comenzar la temporización, se descompondrá en un número binario de 16 bits los 8 bits altos se cargarán en TMR1H y los 8 bits bajos en TMR1L.
P el factor de división del pre-divisor (P = 1, 2, 4, 8).
Ti el periodo de los pulsos internos o externos de entrada al pre-divisor.

Registros asociados al Timer1

Registro	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Ver APENDICE 2
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	A2.5
PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	A2.7
PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	A2.6
TMR1L	Registro que contiene el byte menos significativo del registro TMR1 de 16 bits								
TMR1H	Registro que contiene el byte más significativo del registro TMR1 de 16 bits								
T1CON			T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC#	TMR1CS	TMR1ON	A2.3

FIGURA 3.8: REPRESENTACIÓN EN BLOQUES DEL FUNCIONAMIENTO DEL TIMER 1.

3.2.1 Utilización del Temporizador 1 sin interrupción

Muestra en el display de 7 segmentos (véase la figura del ejercicio 2.2.1) las cifras hexadecimales de 0 al F de manera indefinida. Las cifras cambiarán con una cadencia de 500 ms.

Solución:

La cadencia de cambio de 500 ms se conseguirá con una función de retardo de 500 ms. Dicha función de retardo se realizará con el temporizador 1 sin utilizar las interrupciones del mismo.

Configuración

La temporización o tiempo de desbordamiento del Timer1 es Td = (65536 – TMR1) * P * Ti donde

Ti es el período de los pulsos,

P es el pre-divisor,

TMR1 es el valor con el que hay que cargar el temporizador 1 (TMR1 = TMR1H // TMR1L)

El Timer1 se configura con el registro T1CON (véase apéndice A2.3)

Con T1CON = 0x31 = 0b00110001 el Timer1 actúa como temporizador (Ti = 4/4 MHz = 1 µs) y predivisor P = 8.

Para 500 ms, el valor con el que hay que cargar el temporizador 1 es:

TMR1 = 3036 = 0x0BDC ⇒ TMR1H = 0x0B y TMR1L = 0xDC

Teniendo en cuenta estos valores se realiza la función de retardo DELAY500ms

Código fuente

```
//ARCHIVOS DE DEFINICIONES
#include<htc.h>                         //Incluimos librería del micro a usar
#include "timers.h"                     //Incluimos librería de timers
#include "ports.h"                      //Incluimos librería de puertos
#define _XTAL_FREQ 4000000              //Oscilador Interno de 4MHZ

__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

//DEFINICIÓN DE VARIABLES
//Tabla con el código de 7 segmentos complementado para las cifras de 0 a F.
//(tamaño byte)

unsigned char codigo[]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,
                                        0x90,0x88,0x83,0xC6,0xA1,0x86,0x8E};
unsigned char x;                        //Índice de lista de código

// DEFINICION DE FUNCIONES
//Función de retardo de 500mseg
void DELAY500ms(void){
        set_timer1(0x0BDC);             //Ajusta los registros contadores del Timer1
        setup_timer1(T1_ON|T1_INT|T1_DIV8);     //Timer 1 ON con contador interno
                                                //y predivisor 8
        while(TMR1IF == 0) continue;    //Espera mientras no se desborde el
                                        //Timer 1
        TMR1IF=0;
}
// PROGRAMA PRINCIPAL
void main(void){
        set_all_digital();              //Desactiva PORTA como entradas analógicas
        TRISA = 0b00000000;             //Configura el PORTA como salidas
        TRISD=0x00;                     //Configura el PORTD como salidas
        PORTA=1;                        //Activar display de RA0
        x=0;                            //Índice de la lista para la primera cifra.
        while(1){
           PORTD=codigo[x];             //Saca el valor que corresponde por el puerto D
           DELAY500ms();                //Espera 500 ms
           x++;                         //Incrementa índice de la lista de códigos
```

```
        if(x==16){                          //Si se ha barrido toda la lista
                x=0;                        //Resetear índice de la lista
        }
    }
}
```

Se puede observar el funcionamiento mediante el analizador lógico del simulador.

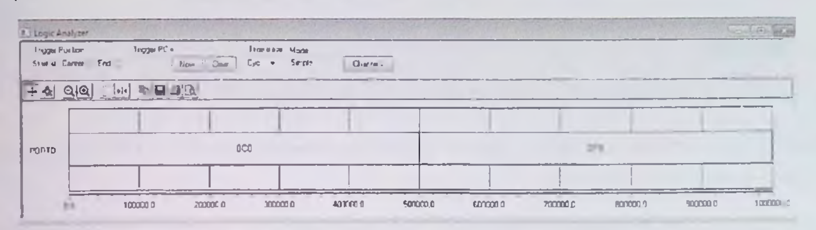
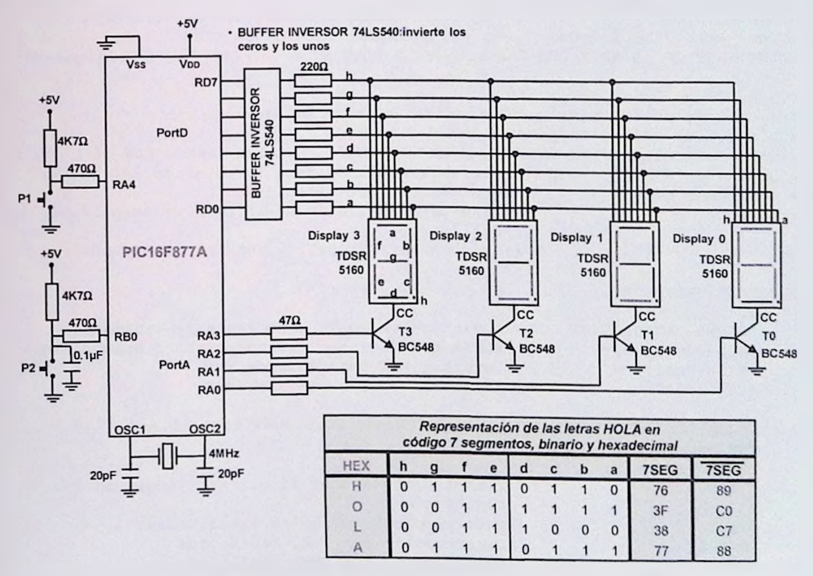


FIGURA 3.9: SIMULACIÓN DEL EJERCICIO 3.2.1.

3.2.2 Utilización del Temporizador 1 con interrupción

Se pretende realizar un letrero con la palabra HOLA que se vaya desplazando de derecha a izquierda con una cadencia de 1 segundo. Se quiere disponer de dos pulsadores: uno para que el letrero empiece a desplazarse (P1) y el otro (P2) para parar el letrero como se muestra en la siguiente figura.



Representación de las letras HOLA en código 7 segmentos, binario y hexadecimal

HEX	h	g	f	e	d	c	b	a	7SEG	$\overline{\text{7SEG}}$
H	0	1	1	1	0	1	1	0	76	89
O	0	0	1	1	1	1	1	1	3F	C0
L	0	0	1	1	1	0	0	0	38	C7
A	0	1	1	1	0	1	1	1	77	88

FIGURA 3.10: CONEXIONES DEL PIC16F877A A LOS 4 DISPLAYS (MULTIPLEXADOS) DEL LETRERO Y A LOS PULSADORES DE MARCHA Y PARADA.

Solución:

Se utilizará el temporizador 1 con interrupciones para conseguir la cadencia de 1 segundo y el temporizador 0 sin interrupciones para conseguir los 5 ms que hay que mantener activado cada display para que no parpadee.

Configuración

El Timer1 se configura con el registro T1CON (véase apéndice A2.3)

Con T1CON=0x31=0b00110001 el Timer1 actúa como temporizador (Ti = 4/4 MHz = 1 µs), con predivisor P=8 y habilitado.

Como no se puede conseguir un tiempo de desbordamiento de 1 segundo, se calcula TMR1 para que sea de 500 ms y con 2 desbordamientos se tendrá 1 segundo. Entonces, para 500 ms, el valor con el que hay que cargar el temporizador 1 es:

$$TMR1 = 3036 = 0x0BDC \Rightarrow TMR1H = 0x0B \text{ y } TMR1L = 0xDC$$

Código fuente

```
//ARCHIVOS DE DEFINICIONES
#include<htc.h>                       //Incluimos librería del micro a usar
#define _XTAL_FREQ 4000000            //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);   //Configuración
                                                                 //del PIC

//Variables globales para que sean accesibles en la interrupción
//Tabla con el código 7segmentos complementado
unsigned char D4=0x89,D3=0xC0,D2=0xC7,D1=0x88;  //código 7segment0s de H O L A
unsigned char AUX;            //variable auxiliar para la rotación del letrero.
unsigned char cont = 0;       //Contador de interrupciones
bit ESTADO;                   //Flag para activar o desactivar el desplazamiento
                              //del letrero
                              //(MARCHA es ESTADO=1) y (PARADA es ESTADO=0)

//PROGRAMA PRINCIPAL
void main(){
        ADCON1=0b00000110;    //Desactivamos PORTA como entradas analógicas
        TRISA=0x10;           //PORTA con RA4 como entrada y resto como salidas
        TRISB=0x01;           //Terminal RB0 como entrada
        TRISD=0x00;           //Terminales PORTD como salidas

//Configuración del TIMER1 para producir interrupciones cada 500ms.
        TMR1H=0x0B;
        TMR1L=0xDC;
        T1CON=0x31;           //TIMER1 temporizador Ti = 1 microseg, con P=8 y
                              //habilitado
        GIE=1;                //Interrupciones Globales habilitadas
        PEIE=1;               //Interrupción por TMR1 habilitada
        TMR1IE=1;             //Interrupción por TMR1 habilitada
```

```
      while(1){
        if(RA4 == 0){          //Se ha presionado la MARCHA
                ESTADO=1;      //Ponemos letrero en MARCHA
        }
        if(RB0 == 0){          //Se ha presionado la tecla de PARADA
                ESTADO = 0; //Ponemos letrero en PARADA
                D4=0x89;
                D3=0xC0;
                D2=0xC7;
                D1=0x88;       //Reseteamos el letrero para empezar HOLA
        }

//Código para mostrar los dígitos
        RA3=0;RA2=0;RA1=0;RA0=1;   //Activamos DISPLAY 0 y los otros desactivados
        PORTD = D1;                //Mostramos por puertoD la letra que hay en D1

        __delay_ms(5);             //Retardo de 5 ms para evitar el parpadeo

        RA3=0;RA2=0;RA1=1;RA0=0;   //Activamos DISPLAY 1 y los otros desactivados
        PORTD = D2;                //Mostramos por puertoD la letra que hay en D2

        __delay_ms(5);             //Retardo de 5 ms para evitar el parpadeo

        RA3=0;RA2=1;RA1=0;RA0=0;   //Activamos DISPLAY 2 y los otros desactivados
        PORTD = D3;                //Mostramos por puertoD la letra que hay en D3

        __delay_ms(5);             //Retardo de 5 ms para evitar el parpadeo

        RA3=1;RA2=0;RA1=0;RA0=0;   //Activamos DISPLAY 3 y los otros desactivados
        PORTD = D4;                //Mostramos por puertoD la letra que hay en D4

        __delay_ms(5);             //Retardo de 5 ms para evitar el parpadeo
      }
}

//INTERRUPCIÓN
static void interrupt isr(){
      if(TMR1IF){                  //Si la interrupción la ha generado TMR1,
                                   //ejecutamos el código
        TMR1IF=0;                  //Desactivamos el FLAG de la interrupción
                                   //de TMR1
        TMR1H=0x0B;
        TMR1L=0xDC;
        if(ESTADO){                //Si el letrero está en marcha
                cont++;            //Incrementamos la cuenta de interrupciones
                if(cont == 2){     //Si se ha completado el segundo = 500ms * 2
                                   //interrupciones
                    cont = 0;      //Reseteamos la cuenta de interrupciones
                    AUX = D4;      // Rotamos las letras
                    D4 = D3;
                    D3 = D2;
                    D2 = D1;
                    D1 = AUX;
                }
        }
      }
}
```

3.2.3 Utilización del Temporizador 1 como contador de pulsos

Se quiere que el led D1 se encienda o se apague cada vez que el led D0 se ha encendido y apagado 30 veces. La cadencia de encendido y apagado del led D0 será de 500 ms.

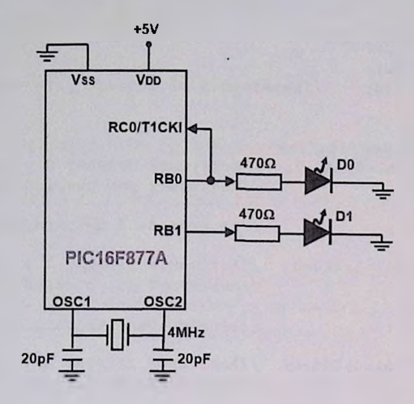


FIGURA 3.11: CONEXIONES DEL PIC16F877A. LOS PULSOS QUE SALEN POR RB0 (GENERADOS POR TIMER0) SON LOS PULSOS EXTERNOS QUE ENTRAN POR T1CKI AL TIMER1.

Solución:

Se utilizará el timer0 para generar los pulsos que encienden y apagan el led D0. Se utilizará el timer1 como contador de los 30 pulsos generados por el timer0 para encender o apagar el led D1.

Configuración

El Timer0 se configura con el registro OPTION_REG (véase apéndice A2.2)

Con OPTION_REG = 0b11000111 = 0xC7 el Timer0 actúa como temporizador (Ti = 4/4 MHz = 1 µs) y con predivisor P = 256.

Para conseguir que el tiempo de desbordamiento (interrupción) sea de 50 ms hay que cargar TMR0 con N_{TMR0} = 60.

Con 10 desbordamientos o interrupciones (controladas por cont) se tendrán 500ms y se cambiara RB0.

El Timer1 se configura con el registro T1CON (véase apéndice A2.3)

Con T1CON = 0x07 = 0b00000111 el Timer1 actúa como contador de pulsos externos que entran por T1CKI, con predivisor P=1 y habilitado.

Los pulsos externos que entran por T1CKI son los pulsos que salen por RB0 producidos por el Timer0 y que son de 1 segundo de periodo.

Para que RB1 (D1) cambie cada 30 segundos, el Timer1 se tendrá desbordar con 30 pulsos para ello hay que cargar el temporizador 1 con:

TMR1 = 65536 – 30 = 65506 = 0xFFE2 ⇒ TMR1H = 0xFF y TMR1L = 0xE2

Código fuente

```
//ARCHIVOS DE DEFINICIONES
#include <htc.h>                          //Incluimos librería del micro a usar
#include "timers.h"                       //Incluimos librería de timers
#include "ports.h"                        //Incluimos librería de puertos
#define _XTAL_FREQ 4000000                //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

//DEFINICIÓN DE VARIABLES
unsigned char cont;                       //Contador de desbordamiento de TMR0

//PROGRAMA PRINCIPAL
void main(void){
      TRISB = 0b00000000;                 //Configura el PORTB como salidas
      TRISC=0xFF;                         //Configura el PORTC como entradas
      PORTB=0;                            //Todos los led apagados

//Configuración del TIMER0 para producir interrupciones cada 50 ms.
      setup_timer0(RTCC_INTERNAL|RTCC_DIV256);     //TIMER0 como temporizador
                                                   //con predivisor P=256
      set_timer0(60);               //Para que Td sean 50ms
      ei();                         //Habilitar interrupciones
      T0IE=1;                       //Habilitar interrupción de TMR0
      T0IF=0;                       //Desactivar el FLAG de TMR0
//Configuración del TIMER1 para producir interrupciones cada 30 pulsos externos
      set_timer1(0xFFE2);
      setup_timer1(T1_ON|T1_EXT|T1_DIV1);     //TIMER1 contador pulsos externos
                                              //por T1CKI, con P=1 y ON
      PEIE=1;                       //Habilitar interrupción de TMR1
      TMR1IE=1;                     //Habilitar interrupción de TMR1
      TMR1IF=0;                     //Desactivar el FLAG de TMR1
      while(1);                     //Bucle infinito.
}
//INTERRUPCIÓN por Timer0 y timer1
static void interrupt isr(void){
      if(T0IF && !TMR1IF){
        T0IF=0;                //Desactivar el FLAG de la interrupción de TMR0
        TMR0=60;               //Recargar TMR0
        cont++;                //Incrementar contador de interrupciones
        if(cont==10){          //10 interrupciones cada 50 ms da lugar a 500 ms
              PORTBbits.RB0 = ~PORTBbits.RB0;    //Cambia estado de RB0
              cont=0;               //Reiniciar contador de desbordamientos
        }
      }
      if(TMR1IF == 1){
        TMR1IF=0;                     //Desactivar el FLAG de la interrupción de TMR1
        set_timer1(0xFFE2);           //Recargar TMR1
        PORTBbits.RB1 = ~PORTBbits.RB1;     //Cambia estado de RB1
      }
}
```

3.2.4 Utilización del Temporizador 1 con un oscilador externo

Se quiere que los LED se enciendan y apaguen cada 5 segundos. Empiezan encendidos D3-D0 y apagados D2-D1. Después de 5 segundos se apagan D3-D0 y se encienden D2-D1 y así sucesivamente.

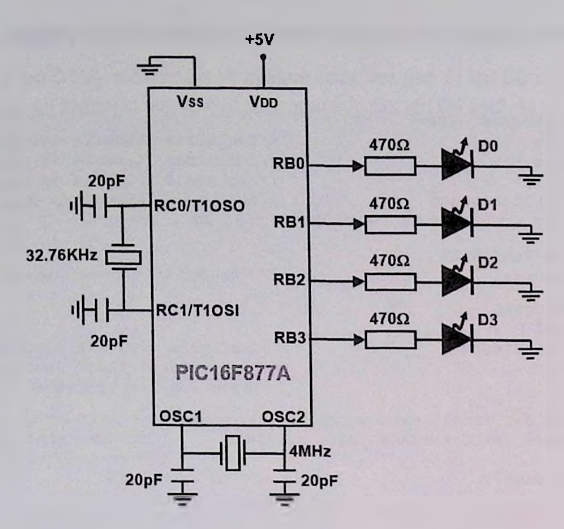


FIGURA 3.12: CONEXIONES CON EL PIC16F877A DEL OSCILADOR EXTERNO DE 32.76 KHZ PARA CONTROLAR EL TIMER1.

Solución:

Se utilizará el Timer1 para generar la temporización de 5 segundos. Se utilizará un oscilador externo de 32.76 KHz y se emplearán interrupciones.

Configuración

El Timer1 se configura con el registro T1CON (véase apéndice A2.3)

Con T1CON = 0x3F = 0b00111111 el Timer1 actúa como contador de pulsos externos, con predivisor P = 8 y habilitado. Los pulsos externos que entran son generados por el oscilador externo de 32.76 KHz y cuyo periodo es de 30.52 µs.

Para que se desborde o se produzca interrupción cada 5 segundos hay que cargar el temporizador 1 con:

$$TMR1 = 65536 - \frac{5s}{8*(1/32.76KHz)} = 45061 = 0xB005 \Rightarrow TM1H = 0xB0 \text{ y } TMR1L = 0x05$$

Con cada interrupción se cambiarán los LED encendidos y apagados.

Código fuente

```
//ARCHIVOS DE DEFINICIONES
#include <htc.h>                  //Incluimos librería del micro a usar
#define _XTAL_FREQ 4000000        //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);
```

```
//PROGRAMA PRINCIPAL
void main(void){
      TRISB = 0b00000000;          //Configura el PORTB como salidas
      PORTB=0b00001001;            //Encendidos D3-D0 y apagados D2-D1

//Configuración del TIMER1 para producir interrupciones cada 5 segundos
      TMR1H=0xB0;
      TMR1L=0x05;
      T1CON=0x3F;      //TIMER1 contador pulsos externos generados por oscilador
                       //externo de 32.76 KHz, con P=8 y habilitado
      GIE=1;           //Habilitar interrupciones
      PEIE=1;          //Habilitar interrupción de TMR1
      TMR1IE=1;        //Habilitar interrupción de TMR1
      TMR1IF=0;        //Desactivar el FLAG de TMR1
      while(1);        //Bucle infinito.
}

//INTERRUPCIÓN por Timer1
static void interrupt isr(void){
      if(TMR1IF){
         TMR1IF=0;          //Desactivar el FLAG de la interrupción de TMR1
         TMR1H=0xB0;        //Recargar TMR1
         TMR1L=0x05;
         PORTB = ~PORTB;    //Cambia estado de los LED
      }
}
```

Se puede observar el funcionamiento mediante el analizador lógico del simulador. Para ello se ha definido un estímulo de tipo reloj por RC0 con un tiempo de subida y bajada idéntico correspondiente a aproximadamente un semiperiodo de la señal de 32.76 KHz.

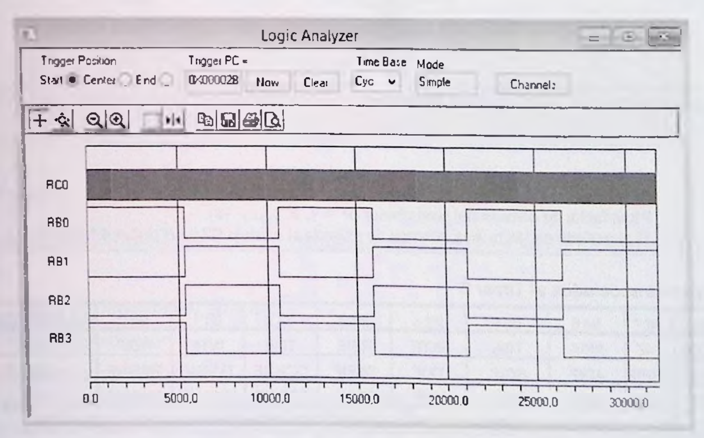


FIGURA 3.13: SIMULACIÓN DEL EJERCICIO 3.2.4.

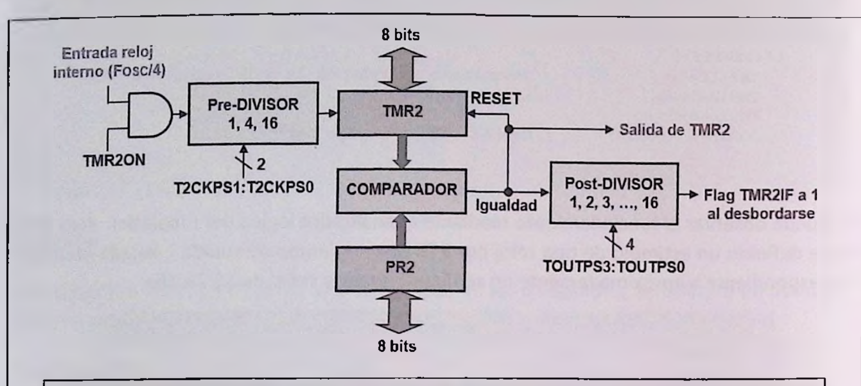
Nota: Para acelerar la simulación se ha ajustado el contador del Timer 1 para una cadencia de 5 ms.

3.3. Temporizador 2

El temporizador 2 es un temporizador de 8 bits con predivisor, postdivisor y registro de período. Usando el predivisor y postdivisor a sus valores máximos el tiempo necesario para que se desborde es el mismo que el de un temporizador de 16 bits.

El temporizador 2 se usa también para establecer la base de tiempo cuando el módulo CCP se utiliza para generar ondas PWM.

El postdivisor cuenta el número de veces que el registro TMR2 coincide con el registro PR2, lo que ayuda a reducir la carga de la CPU.



El tiempo que tarda en desbordarse el Timer2 es:

Temporización del Timer2 = Td = P1 * (N_{PR2}+1) * P2 * T

N_{PR2} el valor que hay que cargar en el registro PR2.
P1 el factor de división del pre-divisor (**P1** = **1, 4, 16**).
P2 el factor de división del post-divisor (**P** = **1, 2, 3, ...,** **16**).
Ti el período de los pulsos internos de entrada al módulo (**Ti = 4/Fosc = 4 * Tosc**)

Registros asociados al Timer 2

Registro	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Ver APENDICE 2
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	A2.5
PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	A2.7
PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	A2.6
TMR2	Registro de módulo de Timer2								
T2CON		TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	A2.4
PR2	Registro de periodo de Timer2								

FIGURA 3.14: REPRESENTACIÓN EN BLOQUES DEL FUNCIONAMIENTO DEL TIMER 2.

3.3.1 Utilización del Temporizador 2 sin interrupción

a) Programa el Timer2, sin interrupción, para obtener por RB3 una señal de 50 Hz. El Timer2 debe activar el flag TMR2IF cada milisegundo, siendo la frecuencia de oscilador principal de 4 MHz

b) Simula el funcionamiento del programa mediante el analizador lógico de *MPLAB SIM*

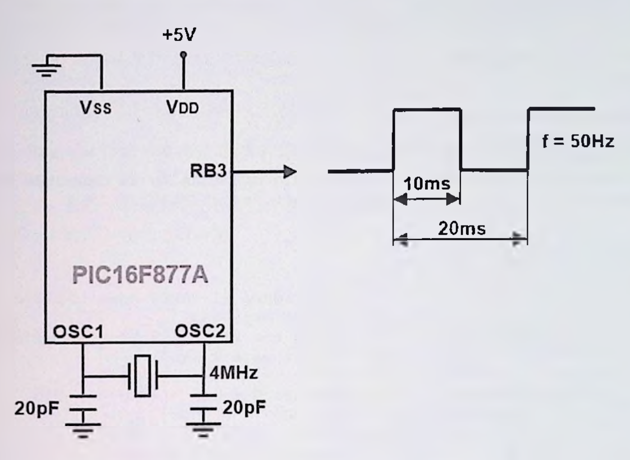


FIGURA 3.15: OBTENER POR EL TERMINAL RB3 UNA ONDA CUADRADA DE 50 HZ.

Solución:

Se utilizará el Timer2 para generar la temporización de 1 ms.

Configuración

La temporización o tiempo de desbordamiento del Timer2 es $Td = P1*(N_{PR2} + 1) * P2 * Ti$ donde

- Ti es el período de los pulsos,
- P1 es el predivisor,
- P2 es el postdivisor
- N_{PR2} es el valor con el que hay que cargar PR2

El Timer2 se configura con el registro T2CON (véase apéndice A2.4)

Con T2CON = 0b01001101 el Timer2 actúa como temporizador (Ti = 4/4 MHz = 1 µs), con P1 = 4 y P2 = 10 y habilitado

Para Td = 1 ms, el valor con el que hay que cargar el PR2 es $N_{PR2} = 24$.

La señal de 50 Hz tiene un período de 20 ms. El semiperiodo de la señal será de 10 ms, es decir, cada 10 desbordamientos de Timer2 (controlados por un contador x) se cambiará el estado del bit en RB3.

Código fuente

```
//ARCHIVOS DE DEFINICIONES
#include<htc.h>                                  //Incluimos librería del micro a usar
#define _XTAL_FREQ 4000000                       //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

//DEFINICION DE VARIABLES
      int x;

//DEFINICION DE FUNCIONES
//Función de 1 ms
void DELAY1ms(void){
      while(TMR2IF == 0);                  //Espera mientras no se desborde el Timer 2
      TMR2IF=0;                            //Desactivar el FLAG de TMR2
}

//PROGRAMA PRINCIPAL
void main(void){
      TRISB=0x00;                          //Configura el PORTB como salidas
      PORTB=0;                             //Nivel bajo RB3
      T2CON = 0b01001101;                  //TMR2 con P1=4 y P2=10 y encendido
      PR2 = 24;                            //Carga para 1 ms
      while(1){
         for(x=0;x<10;x++){                //Espera durante
                DELAY1ms();                //10 ms
         }
         RB3 = ~RB3;                       //Cambiar estado de RB3
      }
}
```

b) Para ver el funcionamiento se puede utilizar el *Stopwatch* del simulador con un *breakpoint*. Se observa que el semiperiodo de la señal es 10.004 ms

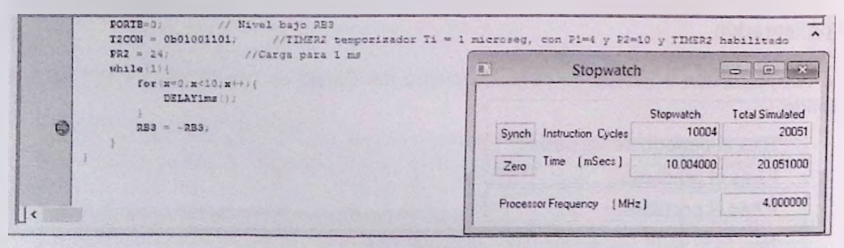


FIGURA 3.16: SIMULACIÓN DEL EJERCICIO 3.3.1 (STOPWATCH).

También se puede observar el funcionamiento mediante el analizador lógico del simulador. El período de la señal es de 20016 ciclos (20016 μs).

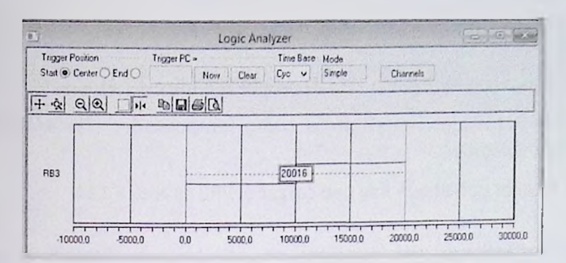


FIGURA 3.17: SIMULACIÓN DEL EJERCICIO 3.3.1 (LOGIC ANALYZER).

3.3.2 Utilización del Temporizador 2 con interrupción

Se pretende realizar un cronómetro digital que cuente segundos y centésimas de segundo, es decir, un cronómetro de 00.00 a 59.99 incrementándose en centésimas de segundo (véase figura 3.18). Se quiere disponer de dos pulsadores: uno (P1) para que funcione el cronómetro y el otro (P2) para pararlo.

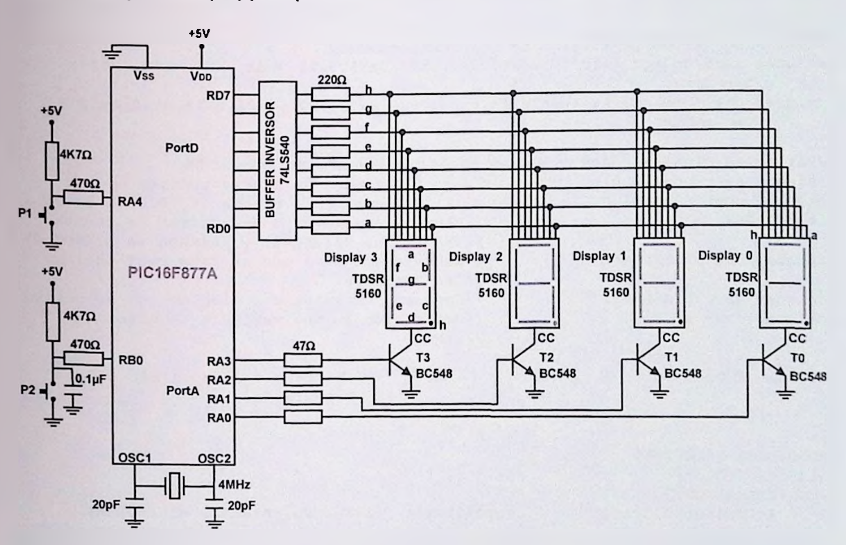


FIGURA 3.18: CONEXIONES DEL PIC16F877A A LOS 4 DISPLAYS (MULTIPLEXADOS) DEL CRONÓMETRO Y A LOS PULSADORES DE MARCHA Y PARADA.

Solución:

Se utilizará el temporizador 2 con interrupciones para conseguir la cadencia de 10 ms (centésima de segundo) que incremente el display 0 y los 5 ms que hay que mantener activado cada display para que no parpadee.

Configuración

El Timer2 se configura con el registro T2CON (véase apéndice A2.4)

Con T2CON = 0b01001101 el Timer2 actúa como temporizador (Ti = 4/4MHz = 1 µs), con P1 = 4 y P2 = 10 y habilitado

Para Td = 5 ms, el valor con el que hay que cargar el PR2 es N_{PR2} = 124.

Con cada desbordamiento (interrupción) se activará un display y con cada dos interrupciones se incrementará el display 0.

Código fuente

```
//ARCHIVOS DE DEFINICIONES
#include<htc.h>                     //Incluimos libreria del micro a usar
#include "timers.h"                 //Incluimos librería de timers
#define _XTAL_FREQ 4000000          //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);   //Configuración
                                                                //del PIC

//Tabla con el código 7segmentos complementado
unsigned char codigo[]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
        //Sin punto
unsigned char codigo1[]={0x40,0x79,0x24,0x30,0x19,0x12,0x02,0x78,0x00,0x10};
        //Con punto

//Variables globales para que sean accesibles en la interrupción
unsigned char cont = 0;                 //Contador de interrupciones
unsigned char S2=0,S1=0,C2=0,C1=0;      //Llevamos la cuenta de 00.00 a 59.99
bit ESTADO;                         //Flag para activar o desactivar la cuenta
                                    //(MARCHA es ESTADO=1) y (PARADA es ESTADO=0)
unsigned char ROTADOR;              //Máscara con el 1 que se rota para activar
                                    //los displays.
unsigned char DISPLAYS;             //Contiene el valor del display que se activa
unsigned char x;                    //definición de una variable 16 bits sin
                                    //signo

#define  ROTA_1I(x) x = (x << 1) | (x >> 7)    //Macro que rota un bit a la
                                               //izquierda.


//PROGRAMA PRINCIPAL
void main(void){
//CONFIGURACIÓN
        ADCON1=0b00000110;      //Desactivamos PORTA como entradas analógicas
        TRISA=0x10;             //PORTA con RA4 como entrada y resto como salidas
        TRISB=0x01;             //Terminal RB0 como entrada
        TRISD=0x00;             //Terminales PORTD como salidas
        ei();                   //Interrupciones Globales habilitadas
        PEIE=1;                 //Interrupción por TMR2 habilitada
        TMR2IE=1;               //Interrupción por TMR2 habilitada
        setup_timer2(T2_ON|T2_PRED_DIV4|T2_POST_DIV10); //TIMER2 con P1=4 y P2=10
                                                        //y encendido
        set_timer2(124);        //Carga para 5 ms
        ESTADO = 0;             //Empieza en PARADO el cronómetro
        ROTADOR = 0x11;         //Máscara con 1 en el bit 0 y 1 en el bit 4.
        DISPLAYS = 0;
```

```
      while(1)
      { if(RA4 == 0)                        //Se ha presionado la MARCHA
        {        ESTADO=1;                  //Ponemos el crono en MARCHA
                 S2=0;S1=0;C2=0;C1=0;       //Reseteamos el cronómetro
        }
        if(RB0 == 0)                        //Se ha presionado la tecla de PARADA
        {        ESTADO = 0;                //Ponemos el crono en PARADA
        }

//Código para mostrar los dígitos
        DISPLAYS = ROTADOR & 0x0F;
        if(DISPLAYS==1)
        {RA3=0;RA2=0;RA1=0;RA0=1;           //Se activa DISPLAY 0 y los otros
                                            //desactivados
        PORTD = codigo[C1];                 //Se muestra por puertoD las centésimas
                                            //de segundo
        }
        if(DISPLAYS==2)
        {RA3=0;RA2=0;RA1=1;RA0=0;           //Se activa  DISPLAY 1 y los otros
                                            //desactivados
        PORTD = codigo[C2];                 //Se muestra por puertoD las décimas
                                            //de segundo
        }
        if(DISPLAYS==4)
        {RA3=0;RA2=1;RA1=0;RA0=0;           //Se activa  DISPLAY 2 y los otros
                                            //desactivados
        PORTD = codigo1[S1];                //Se muestra por puertoD las unidades
                                            //de segundo
        }
        if(DISPLAYS==8)
        {RA3=1;RA2=0;RA1=0;RA0=0;           //Se activa DISPLAY 3 y los otros
                                            //desactivados
        PORTD = codigo[S2];                 //Se muestra por puertoD las decenas
                                            //de segundo
        }
      }
}

//ATENCIÓN A LAS INTERRUPCIONES
static void interrupt isr()
{
      if(TMR2IF){                     //Comprueba si es la interrupción del TMR2
          TMR2IF=0;                   //Desactiva el FLAG de la interrupción de TMR2
        ROTA_1I (ROTADOR);            //Cada 5 ms rota la máscara que activa los
                                      //displays
        if(ESTADO){                   //Si el cronómetro está en marcha
                cont++;               //Incrementa la cuenta de interrupciones
                if(cont == 2){        //Comprueba si han transcurrido 2
                                      //interrupciones (10 ms)
                 cont = 0;            //Resetea la cuenta de interrupciones
                 C1++;                //Incrementa las centésimas de segundo
                 if(C1 == 10){        //Si han llegado a 10 centésimas de segundo
                      C1 = 0;         //Resetea las centésimas de segundo
                      C2++;           //Incrementa las décimas de segundo
                      if(C2 == 10){        //Si han llegado a 10 décimas de segundo
                            C2 = 0;        //Resetea las décimas de segundo
                            S1++;          //Incrementa las unidades de segundo
                            if(S1 == 10){       //Si han llegado a 10 unidades
                                                //de segundo
                                  S1 = 0;       //Resetea las unidades segundo
```

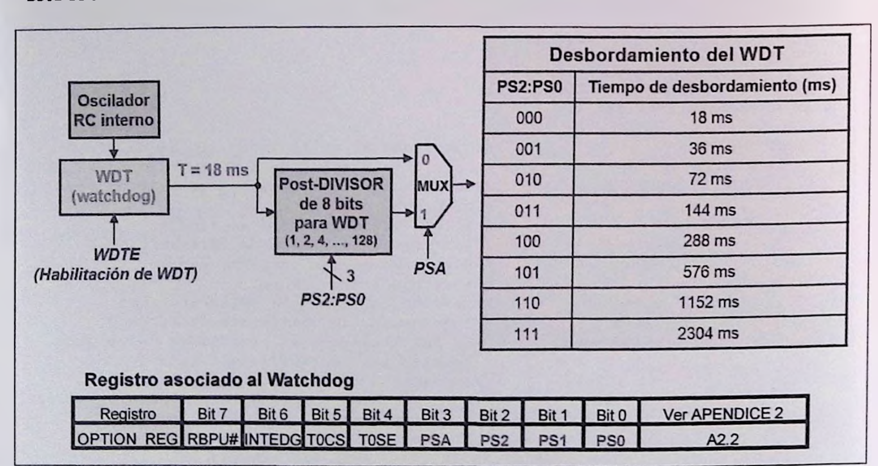
```
                                          S2++;          //Incrementa las decenas de
                                                         //segundo
                                          if(S2 == 6){         //Si han llegado a 6 decenas
                                                               //de segundo
                                                S2 = 0;        //Resetea las decenas de
                                                               //segundo
                                          }
                                    }
                              }
                        }
                      }
             }
      }
}
```

3.4. WATCHDOG

La función del Watchdog es impedir que el sistema entre en un bucle infinito, provocando un RESET cuando esto sucede. El Watchdog es un oscilador RC interno independiente del oscilador principal del microcontrolador, por lo que funciona incluso cuando se detiene la ejecución mediante la instrucción SLEEP.

Se habilita/deshabilita mediante el bit de configuración WDTE. No puede ser deshabilitado por *software*. Poniendo el bit PSA a "1" se habilita el postdivisor del watchdog, mediante el cual se pueden establecer períodos entre 0.018 y 2.30 segundos.

Cuando el Watchdog se desborda, se resetea el microcontrolador, por lo que antes de que esto suceda se debe borrar mediante la instrucción CLRWDT.



Desbordamiento del WDT	
PS2:PS0	Tiempo de desbordamiento (ms)
000	18 ms
001	36 ms
010	72 ms
011	144 ms
100	288 ms
101	576 ms
110	1152 ms
111	2304 ms

Registro asociado al Watchdog

Registro	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Ver APENDICE 2
OPTION_REG	RBPU#	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	A2.2

FIGURA 3.19: REPRESENTACIÓN EN BLOQUES DEL FUNCIONAMIENTO DEL WATCHDOG.

3.4.1 Uso de TIMER 0, TIMER 1 y WATCHDOG

Se desea realizar un programa que, utilizando el pulsador conectado a RA4 y los LED conectados a RB0, RB1, RB2 y RB3 (véase figura 3.20) haga lo siguiente:

Al principio, aparecen todos los led apagados y así continúan hasta que se actúa sobre el pulsador, en ese momento se encienden todos los led y se produce una intermitencia de manera que cada 0,5 s los LED conectados a RB3 y RB2 se encienden y los LED conectados a RB1 y RB0 se apagan y esto durante un total de 10 s, al cabo de los cuales se retorna al estado inicial.

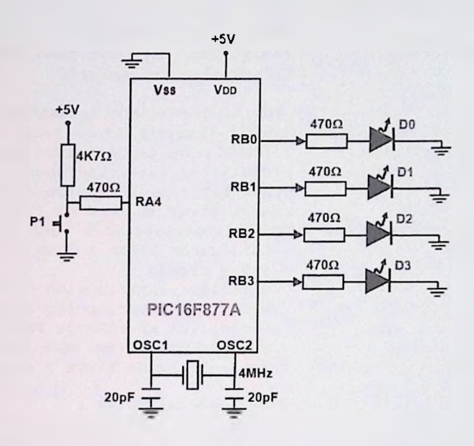


FIGURA 3.20: CONEXIONES CON EL PIC16F877A DE LOS LEDS Y PULSADOR NECESARIO.

Solución:

Se temporizan los 0,5 s de parpadeo con el TIMER 0 y un contador CONT0. Se configura el TIMER0 como temporizador (contador de pulsos del oscilador de 4 MHz), con un pre-divisor de 256 y se carga con NTMR0 = 60 para obtener el desbordamiento o interrupción al cabo de 50 ms. Cargando un contador CONT0 con 10 que se decrementa con cada interrupción, al cabo de 500 ms = 0,5 s valdrá 0.

Se temporizan los 10 s de funcionamiento con el TIMER 1 y un contador CONT1. Se configura el TIMER 1 como temporizador (contador de pulsos del oscilador de 4 MHz), con un predivisor de 8 y se carga con NTMR1 = 3036 = BDCh para obtener el desbordamiento o interrupción al cabo de 500 ms. Cargando un contador CONT1 con 20 y que se decrementa con cada interrupción, al cabo de 10 s valdrá 0.

La vuelta al estado inicial se consigue dejando que el *Watchdog* se desborde (al cabo de 18 ms ya que el postdivisor está asignado al TIMER 0) y resetee al microcontrolador.

Código fuente

```
#include<htc.h>                        //Incluimos librería del micro a usar
#define _XTAL_FREQ 4000000             //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_ON & PWRTE_OFF & FOSC_XT & LVP_OFF);

//DEFINICIÓN DE VARIABLES
unsigned char cont0;                   //Contador de desbordamiento de TMR0
unsigned char cont1;                   //Contador de desbordamiento de TMR1
unsigned char MASCARA;                 //Máscara de leds encendidos y apagados

//PROGRAMA PRINCIPAL
void main(void){
      CLRWDT();                        //Reset del Watchdog
      TRISB = 0b00000000;              //Configura el PORTB como salidas
      PORTB=0x00;                      //Todos los led apagados

      if(RA4 == 0){                    //Se ha presionado la MARCHA
//Configuración del TIMER 0 para producir interrupciones cada 50 ms.
        OPTION_REG=0xC7;               //TIMER0 como temporizador con predivisor P=256
        GIE=1;                         //Habilitar interrupciones
        T0IE=1;                        //Habilitar interrupción de TMR0
        T0IF=0;                        //Desactivar el FLAG de TMR0
//Configuración del TIMER1 para producir interrupciones cada 500 ms.
        T1CON=0x30;                    //Configurar TIMER 1 como temporizador, con
                                       //P=8 y parado
        PEIE=1;                        //Habilitar interrupción de TMR1
        TMR1IE=1;                      //Habilitar interrupción de TMR1
        TMR1IF=0;                      //Desactivar el FLAG de TMR1
        TMR0=60;                       //Para que Td de TMR0 sean 50ms y empiece el TMR0
        TMR1H=0x0B;                    //Para que Td de Timer 1 sean 500 ms.
        TMR1L=0xDC;
        TMR1ON=1;                      //Activado el TIMER 1
        PORTB=0xFF;                    //Todos los Leds encendidos.
        MASCARA=0b11000011;            //RB3-RB2 apagados y RB1-RB0 encendidos.
        while(1){
               CLRWDT();               //Reset del Watchdog
               }
      }
}
//INTERRUPCIÓN por Timer0 y timer1
static void interrupt isr(void){
      if(T0IF && !TMR1IF){
        T0IF=0;                        //Desactivar el FLAG de la interrupción de TMR0
        TMR0=60;                       //Recargar TMR0
        cont0++;                       //Incrementar contador de interrupciones de
                                       //Timer 0.
        if(cont0==10){                 //10 interrupciones cada 50 ms da lugar a
                                       //500 ms
               MASCARA = ~MASCARA;
               PORTB = MASCARA;              //Cambia estado de RB0
               cont0=0;                      //Reiniciar contador de desbordamientos
        }
      }
      if(TMR1IF == 1){
        TMR1IF=0;                //Desactivar el FLAG de la interrupción de TMR1
        TMR1H=0x0B;              //Recargar TMR1
        TMR1L=0xDC;
        cont1++;                 //Incrementar contador de interrupciones de Timer 1.
        if(cont1==20){           //20 interrupciones cada 500 ms da lugar a 10s
               TMR1ON=0;         //Desactivar el TIMER
```

```
                GIE=0;            //Deshabilitar interrupciones
                while(1);         //Bucle infinito

//Como ya trascurrieron los 10 segundos, nos quedamos en este bucle infinito
//sin resetear el Watchdog hasta que desborde y reinicie el microcontrolador.
//Tras 18 mseg (valor típico) el micro se reseteará por desbordamiento del Watchdog
          }
       }
}
```

Comprobamos el funcionamiento mediante el cronómetro del simulador:

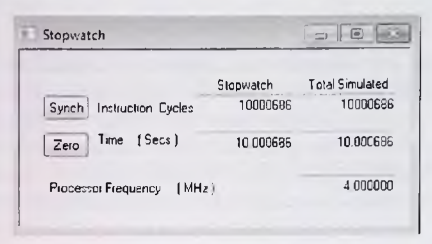


FIGURA 3.21: SIMULACIÓN DEL EJERCICIO 3.4.1 (STOPWATCH).

4. Módulos de Captura, Comparación y Modulación de Anchura de Pulsos (PWM)

4.1. Módulo Captura
4.2. Módulo Comparación
4.3. Módulo PWM

El microcontrolador PIC16F877A contiene dos módulos de captura/comparación y modulación de anchura de pulsos. Estos módulos denominados CCP1 y CCP2 funcionan de forma idéntica con la única excepción del evento especial asociado al funcionamiento en modo comparación. Ambos módulos tienen asociados dos registros de 8 bits CCPRLx (bit bajo) y CCPRHx (bit alto), un registro para su configuración CCPRxCON (A2.9), los bits que permiten a los módulos trabajar mediante interrupciones y el terminal asociado al módulo RC1 para el módulo CCP2 y RC2 para el módulo CCP1.

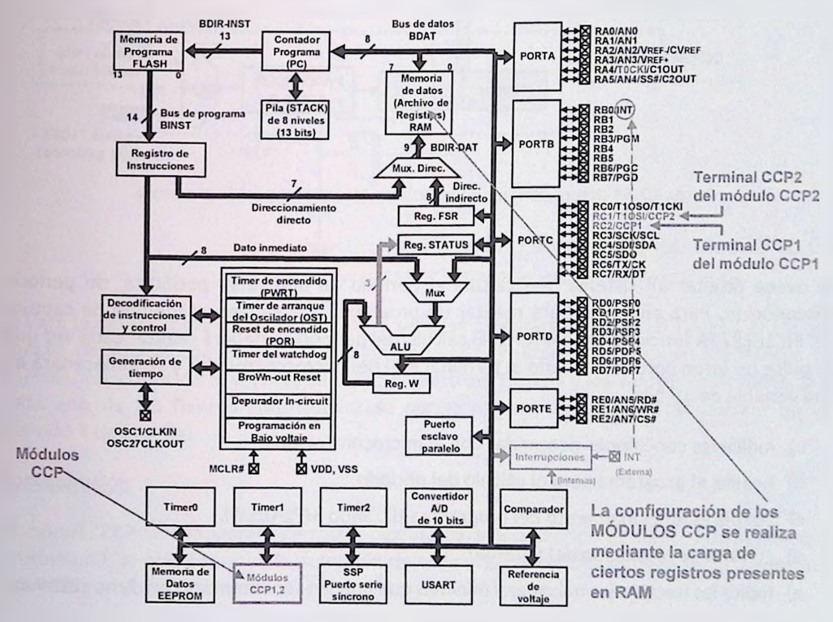


FIGURA 4.1: LOS MÓDULOS CCP DEL MICROCONTROLADOR PIC16F877A.

4.1. Módulo Captura

Los módulos CCP trabajando en modo captura permiten que al producirse un evento se almacene el valor de 16 bits del contador asociado al Timer 1 en los registros CCPRxL y CCPRxH del módulo CCPx. Los eventos a detectar estarán relacionados con la señal en el terminal de entrada del módulo CCPx a elegir entre un flanco de bajada, un flanco de subida, cuatro flancos de subida o 16 flancos de subida. El valor almacenado en los registros del módulo permanecerá hasta que se produzca un nuevo evento.

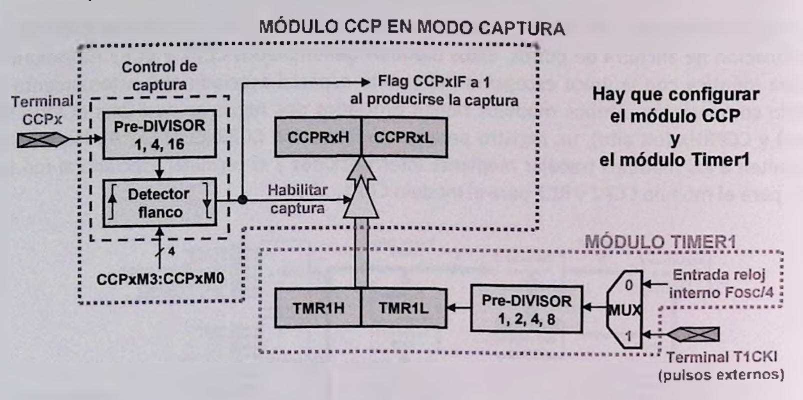


FIGURA 4.2: REPRESENTACIÓN EN BLOQUES DEL FUNCIONAMIENTO DEL MÓDULO CAPTURA.

4.1.1. Cálculo del período de una señal

Se desea diseñar un sistema que calcule el período de una señal periódica, de período desconocido. Para ello se deberá realizar un programa que emplee el módulo de captura del PIC16F877A funcionando a 4 MHz. El cálculo del período se deberá realizar cada vez que se pulse un interruptor conectado al terminal RB0 del microcontrolador y se almacenará en una variable de 16 bits.

a) Indica las conexiones necesarias con el microcontrolador.
b) Realiza el programa para el cálculo del período.
c) Simula el funcionamiento del programa utilizando *MPLABSIM*.
d) ¿Cuál es la resolución del sistema?
e) Indica las frecuencias máximas y mínimas que pueden ser obtenidas con dicho sistema.

Solución:

a) Para el funcionamiento del microcontrolador es necesario conectar correctamente las entradas V_{DD} y V_{SS} a +5 V y 0 V respectivamente. Además, será necesario conectar el

oscilador, en este caso un oscilador de cristal a 4 MHz a las entradas OSC1 y OSC2 tal y como aparece en la figura inferior.

El interruptor se conectará a una entrada digital del microcontrolador. En este caso se ha utilizado la entrada RB0, que se corresponde con la entrada de interrupción externa del microcontrolador, lo que permitirá al programa funcionar en modo interrupción en caso de ser necesario.

La utilización del módulo captura implica que la señal periódica deberá estar conectada a uno de los terminales asociados a estos módulos. En este caso se utilizará el módulo CCP1 por lo que la señal de entrada se conectará al terminal RC2.

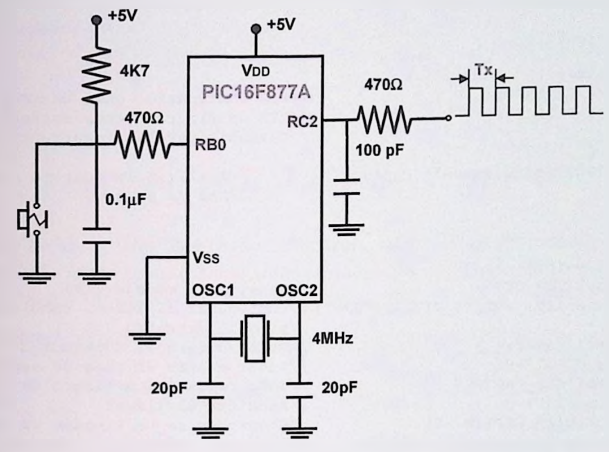


FIGURA 4.3: ESQUEMA DE CONEXIONES DEL PIC16F877A PARA DETERMINAR EL PERIODO DE UNA SEÑAL.

b) Para la determinación del período de la señal se capturará el valor del TIMER1 entre dos flancos de subida consecutivos (también se puede realizar la captura en los flancos de bajada). Así, conocido el valor de cada incremento del TIMER1 y los valores capturados para cada uno de los flancos consecutivos se puede obtener el tiempo transcurrido en un periodo T de la señal.

Configuración

El módulo CCP se configurará en modo captura para la detección de flancos de subida con predivisor 1, lo que permitirá la detección de cada uno de los flancos de la señal de entrada por RC2.

El Timer 1 asociado al modo captura del módulo CCP se configura como temporizador con predivisor 1 por lo que se incrementará cada 4 ciclos de reloj (1 ciclo de instrucción), es decir, cada microsegundo.

Código fuente

```
#include <htc.h>
#include <stdio.h>
#include "ccp.h"
#include "timers.h"

__CONFIG(FOSC_HS & WDTE_OFF & LVP_OFF & PWRTE_ON);
#define _XTAL_FREQ 4000000

typedef union {
    unsigned char bytes[2];
    unsigned int entero;} ubyte2;

    ubyte2 Periodo1,Periodo2;

void Captura(void);

void main(void){
    TRISBbits.TRISB0=1;                    //RB0 configurado como entrada
    TRISCbits.TRISC2=1;                    //RC2 configurado como entrada
    di();                                  //Deshabilitar las interrupciones
    while(1){
      while(PORTBbits.RB0==0) Captura();        //Realizar la captura mientras
                                                //RB0 no pulsado
      }
}

void Captura(void){
    setup_ccp1(CCP_OFF);                   //Resetea el módulo CCP1
    setup_timer1(T1_INT|T1_DIV1|T1_OFF);   //Configurar el Timer1 como temp,
                                           //pred=1, detenido
    set_timer1(0x00);                      //Poner a cero el contador del Timer1
    PIR1bits.CCP1IF=0;                     //Poner a cero el flag de captura
    setup_ccp1(CCP_CAPTURE_RE);            //Modo captura con flanco de subida
    start_timer1();                        //Arrancar el Timer1
    while(PIR1bits.CCP1IF==0);             //Comprobar si ha llegado un flanco
    Periodo1.entero=get_ccp1(value);       //Copia el valor de la captura
    PIR1bits.CCP1IF=0;                     //Baja el flag de captura
    while(PIR1bits.CCP1IF==0);             //comprobar si ha llegado un nuevo
                                           //flanco
    Periodo2.entero=get_ccp1(value);       //Copia el valor de la captura
    Periodo1.entero=Periodo2.entero-Periodo1.entero;   //calcula la diferencia
    setup_ccp1(CCP_OFF);                               //Resetea el módulo CCP1
    stop_timer1();                                        //Detiene el Timer1
    printf("Periodo=%d microsegundos\n",Periodo1.entero); //Muestra el periodo
}

void putch(unsigned char byte){
    TXSTA=0x26;
    //RCSTA|=0x80; //
    RCSTAbits.SPEN=1;
    TXREG=byte;
    while(!TXIF)continue;
    TXIF=0;
}
```

c) Para la simulación del programa del apartado b) será necesario seleccionar la herramienta *MPLABSIM* del menú *debugger* y ajustar la frecuencia del oscilador a 4MHz (menú *debugger / settings*).

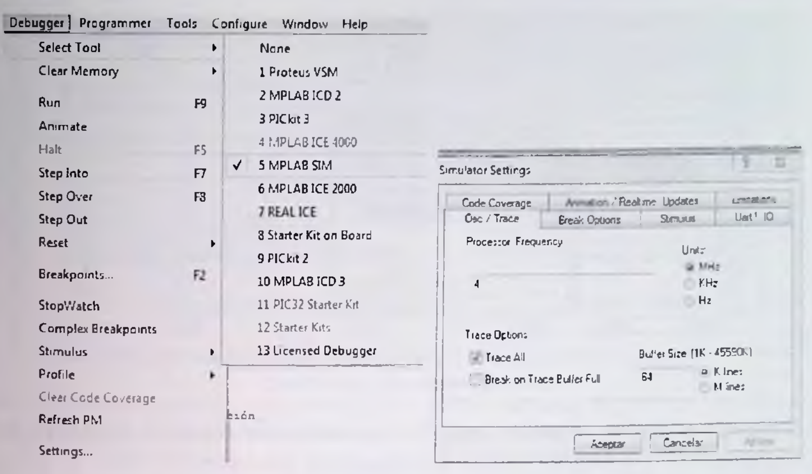


FIGURA 4.4: CONFIGURACIÓN DE MPLAB SIM PARA SIMULAR EL EJERCICIO 4.1.1.

La generación de las señales de entrada se realizará mediante el menú *debugger / Stimulus / New Workbook* donde se creará una señal periódica de tipo *Clock Stimulus* por RC2 y una señal asíncrona que simulará el pulsador conectado a RB0.

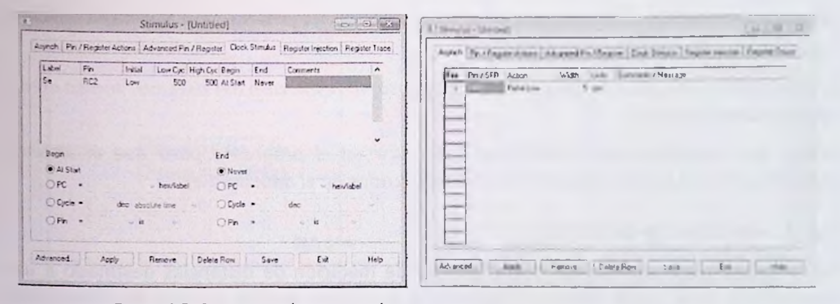


FIGURA 4.5: CONFIGURACIÓN DE LOS ESTÍMULOS EXTERNOS PARA SIMULAR EL EJERCICIO 4.1.1.

Si se ajusta una señal de entrada con período 1 ms a la entrada de RC2 se obtiene que tras activar el pulsador RB0 se ha guardado en la variable Periodo1 el valor 0x03E8 (1000 en decimal) que es el período de la señal en microsegundos.

Una vez ejecutado el programa se pueden observar las señales en RB0 y RC2 mediante el analizador lógico (menú *View/Logic Analyzer*), el mensaje que indica el período en la pestaña SIM Uart1 de la ventana output y el valor de la variable en la ventana *Watch* (menú *View/Watch*).

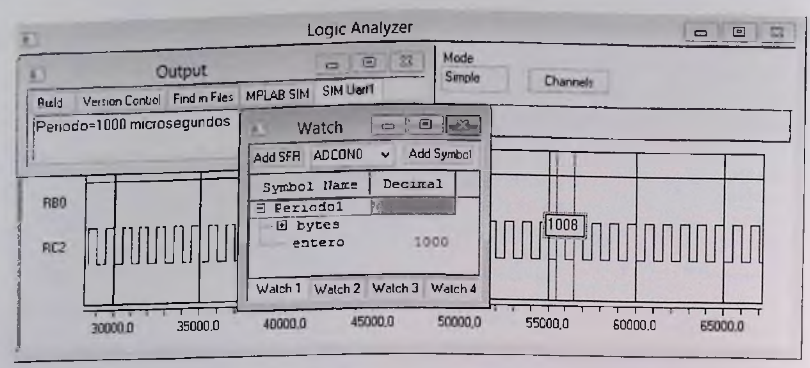


FIGURA 4.6: SIMULACIÓN DEL EJERCICIO 4.1.1.

d) Dado que el período se calcula según los incrementos del TIMER1 y el TIMER1 está configurado para incrementarse cada µs, la resolución del sistema será de ±1µs.

e) Las frecuencias máximas y mínimas del sistema vendrán determinadas por el intervalo de tiempo almacenado en la variable periodo2 que se corresponden con los incrementos en microsegundos capturados del TIMER1. Así pues, los valores mínimo (0x0000) y máximo (0xFFFF) corresponderán a las frecuencias de 1 MHz y 15.259 Hz respectivamente.

Modifica el programa del apartado b) para que funcione mediante las interrupciones asociadas a RB0 y al módulo CAPTURA.

¿Sería posible modificar el programa para mejorar la resolución del sistema? Indica cómo, en caso de ser posible.

Indica qué modificaciones habría que introducir en el programa para que el sistema pudiera capturar frecuencias menores a las indicadas en el apartado e).

4.1.2. Medidor de distancias

Se ha desarrollado un sistema digital portátil de medición de distancias destinado a uso inmobiliario. Para medir distancias (figura 4.7), el sistema emitirá un tono de ultrasonidos (f > 20 KHz) a través un emisor E; esta señal rebotará en la pared opuesta cuya separación se desea medir y será detectada por el sistema a través del receptor R. Calculando el tiempo que tarda la onda en su viaje de ida y vuelta, y teniendo en cuenta que la velocidad del sonido en el aire (a 20 °C es aproximadamente 340 m/s), el sistema puede estimar la distancia entre ambas paredes.

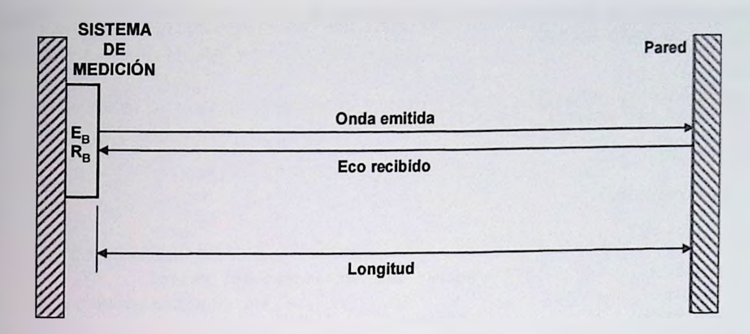


FIGURA 4.7: FUNCIONAMIENTO DEL SISTEMA MEDIDOR DE DISTANCIAS DEL EJERCICIO 4.1.2.

Para la implementación de este sistema (figura 4.8) se emplea un PIC16F877A, en concreto **su módulo CCP1 funcionando en modo captura.**

Cuando se pulsa el pulsador P el emisor emite el tono y a la vez empieza a funcionar el Timer1 del PIC. Cuando el receptor recibe el tono, genera un impulso que se aplica al terminal CCP1 del módulo CCP1 y se captura el valor del Timer1.

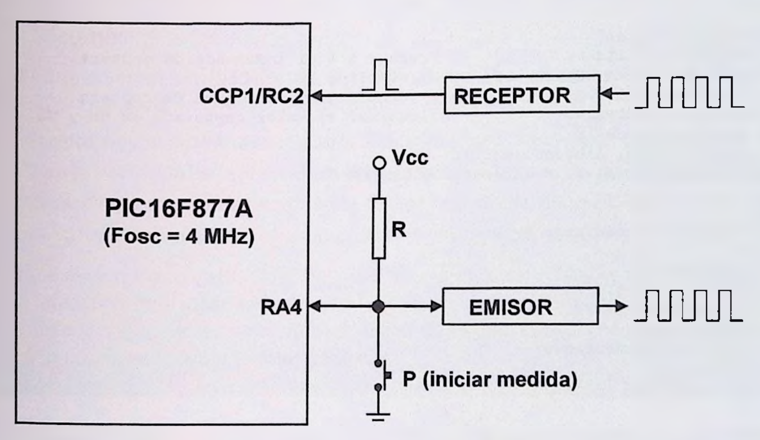


FIGURA 4.8: CONEXIONES DEL SISTEMA MEDIDOR DE DISTANCIAS DEL EJERCICIO 4.1.2 CON EL MICROCONTROLADOR.

A continuación, se muestra el listado del programa que controla el sistema de medición.

Código fuente

```
#include <htc.h>
#include <stdio.h>
```

```
__CONFIG(FOSC_HS & WDTE_OFF & LVP_OFF & PWRTE_ON);
#define _XTAL_FREQ 4000000

typedef union {
    unsigned char bytes[2];
    unsigned int entero;} ubyte2;

    ubyte2 N;

void Captura(void);

void main(void){

    TMR1H=0;                    //Borrar los contadores del Timer1
    TMR1L=0;
    T1CON=0x20;                 //Configurar el Timer1
    CCP1CON=0;                  //Resetear el módulo CCP1
    TRISA=0X10;                 //Configurar RA4 como entrada
    TRISCbits.TRISC2=1;         //Configurar RC2 como entrada
    PIE1bits.TMR1IE=0;          //Inhabilitar interrupción del Timer1
    PIE1bits.CCP1IE=0;          //Inhabilitar interrupción del módulo CCP1.
    PIR1=0;                     //Poner a 0 las banderas (flags) de interrupción.
    CCP1CON=0X04;               //Configurar el módulo CCP1 en modo captura, flanco
                                //de bajada en RC2
    while(PORTAbits.RA4==1);    //Esperar a que baje RA4
    T1CONbits.TMR1ON=1;         //Iniciar el conteo del TMR1
    Captura();
}

void Captura(void){
    PIR1bits.CCP1IF=0;              //Poner a 0 el indicador de captura.
    while(PIR1bits.CCP1IF==0);      //¿CCP1IF = 1?
    PIR1bits.CCP1IF=0;              //poner a 0 el indicador de captura
    N.bytes[0]=CCPR1L;              //guardar el valor capturado en NH y NL.
    N.bytes[1]=CCPR1H;
    printf("N=%d, distancia=%5.3f
m\n",N.entero,4*(float)N.entero*0.000001*340/2);
}

void putch(unsigned char byte)
{
    TXSTA=0x26;
    //RCSTA|=0x80; //
    RCSTAbits.SPEN=1;
    TXREG=byte;
    while(!TXIF)continue;
    TXIF=0;
}
```

Se pide:

a) Calcula la distancia entre las paredes si al final de la captura se tiene en N el valor 40E2h

b) Calcula la máxima distancia que se podrá medir con dicho sistema.

<u>Solución:</u>

a) Calcula la distancia entre las paredes si al final de la captura se tiene en N el valor 40E2h

El Timer1 empieza con sus registros a cero, con predivisor P = 4 y como temporizador cuenta impulsos de duración Ti = 4/Fosc = 4/4 MHz = 1 µs.

La captura se hace con el primer impulso que entra por RC2, luego el tiempo empleado por el tono en recorrer 2 veces la longitud L es:

$$\text{Tiempo} = N * P * Ti = 40E2h * 4 * 1\mu s = 66440\ \mu s$$

Luego:

$$2L = \text{velocidad del sonido} * \text{Tiempo} \Rightarrow 2L = 340\ m/s * 66{,}44\ ms \Rightarrow L = 11{,}29\ m$$

b) Calcula la máxima distancia que se podrá medir.

b.1) Si consideramos la máxima cuenta N = FFFFh tenemos:

$$\text{Tiempo} = N * P * Ti = FFFFh * 4 * 1\ \mu s = 262140\ \mu s$$
$$2L = \text{velocidad del sonido} * \text{Tiempo} \Rightarrow 2L = 340\ m/s * 262{,}14\ ms \Rightarrow L = 44{,}56\ m$$

b.2) Si consideramos la máxima cuenta N = FFFFh y además configuramos el predivisor de Timer1 con el mayor valor P = 8 tenemos:

$$\text{Tiempo} = N * P * Ti = FFFFh * 8 * 1\ \mu s = 524280\ \mu s$$
$$2L = \text{velocidad del sonido} * \text{Tiempo} \Rightarrow 2L = 340\ m/s * 524{,}28\ ms \Rightarrow L = 89{,}12\ m$$

4.1.3. Medidor de anchura de pulsos

Se pide implementar un medidor de distancias basado en un sistema microcontrolador PIC16F877A trabajando a 4 MHz. Especificaciones del sistema:

- El cálculo de la distancia se realizará mediante la activación de un pulsador.
- Para la obtención de la distancia se utilizará el módulo de captura del microcontrolador funcionando en modo interrupción combinado con el dispositivo HC-SR04.
- La distancia calculada en cm se guardará en una variable de 16 bits.

El módulo de ultrasonidos HC-SR04 (http://www.micropik.com/PDF/HCSR04.pdf) se utiliza para la medición de distancias entre 2 cm y 400 cm y se basa en la combinación de un emisor (altavoz) y un receptor (micrófono). Dicho dispositivo cuenta con cuatro conexiones, como se muestra en la figura inferior izquierda.

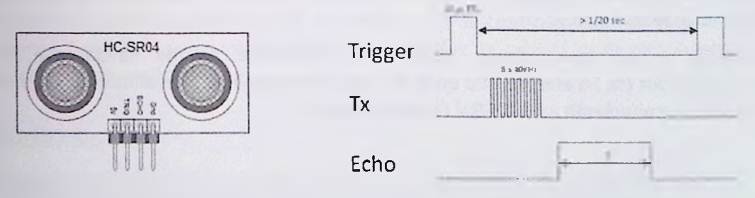


FIGURA 4.9: REPRESENTACIÓN ESQUEMÁTICA Y FUNCIONAMIENTO DEL SENSOR DE ULTRASONIDOS HC-SR04.

El funcionamiento, como se observa en la figura superior derecha, consiste en tres sencillos pasos.

1. Activación de la entrada *trigger* del dispositivo durante 10 μs.
2. Emisión de 8 pulsos a 40 KHz.
3. Generación de un pulso por la salida *echo* proporcional a la distancia.

La duración del pulso será proporcional a la distancia recorrida por el sonido en el viaje de ida y vuelta y vendrá dada por la fórmula:

$$D = \frac{(T*340)}{2}$$

Donde T es la duración del pulso en segundos y 340 m/s es la velocidad aproximada del sonido en el aire a 20 °C.

Se pide:

a) Indica las conexiones con el microcontrolador
b) Realiza el programa para el cálculo de la distancia.
c) Simula el funcionamiento del programa utilizando *MPLABSIM*
d) Indica la resolución del sistema propuesto.

Solución:

a) Para el funcionamiento del microcontrolador es necesario conectar correctamente las entradas V_{DD} y V_{SS} a +5 V y 0 V respectivamente. Además, como el PIC16F877A no cuenta con oscilador interno, será necesario conectar el oscilador (en este caso un oscilador de cristal a 4 MHz) a las entradas OSC1 y OSC2 tal y como aparece en la figura 4.10.

El interruptor se conectará a una entrada digital del microcontrolador. En este caso se ha utilizado la entrada RB0, que permitirá el funcionamiento mediante interrupción.

La utilización del módulo de captura implica que la señal generada por el módulo HC-SR04 (*echo*) deberá estar conectada a uno de los terminales asociados a estos módulos (RC1/RC2). En este caso se utilizará el módulo CCP1 por lo que la señal de entrada se conectará al terminal RC2 (también se podría haber utilizado el módulo CCP2 conectando la salida *echo* al terminal RC1.

La entrada *trigger* del módulo HC-SR04 se conectará a una salida digital del microcontrolador (se ha elegido RC0 en este caso) y los terminales de alimentación y tierra del módulo se conectarán a +5 V y 0 V respectivamente.

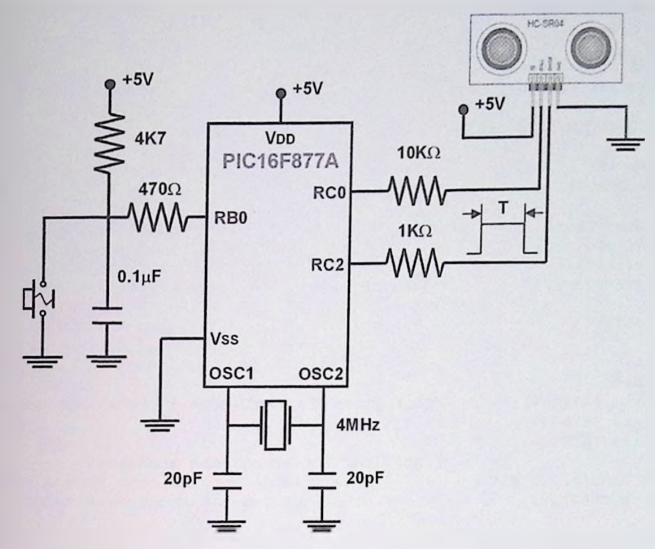


FIGURA 4.10: CONEXIONES DEL SENSOR DE ULTRASONIDOS HC-SR04 CON EL MICROCONTROLADOR PARA EL EJERCICIO 4.1.3.

b) Para la determinación de la anchura del pulso se capturará el valor del TIMER1 entre un flanco de subida y un flanco de bajada. Así, conocido el valor de cada incremento del TIMER1 y los valores capturados para cada uno de los flancos se puede obtener la duración del pulso.

Configuración

El terminal RB0 se configurará para funcionar mediante interrupción por flanco de subida (por defecto) y se encargará de la generación de los pulsos por la salida RC0 (*trigger)*.

El módulo CCP se configurará en modo captura para la detección de flancos de subida con predivisor 1 lo que permitirá la detección del primer flanco de subida y a continuación se configurará para la detección de flancos de bajada permitiendo detectar el final del pulso por RC2.

El Timer 1 asociado al modo captura del módulo CCP se configura como temporizador con predivisor 1 por lo que se incrementará cada 4 ciclos de reloj (1 ciclo de instrucción), es decir, cada microsegundo.

Código fuente

```
#include <htc.h>
#include <stdio.h>

__CONFIG(FOSC_HS & WDTE_OFF & LVP_OFF & PWRTE_ON);
```

```
#define _XTAL_FREQ 4000000

typedef union {
    unsigned char bytes[2];
    unsigned int entero;} ubyte2;

    ubyte2 Dist, DistTemp;

float Distancia;
unsigned char Aux;

void Pulsador(void);
void Flanco(void);
void Tempor(void);
void Trigger(void);

void main(void){
    Aux=0;
    T1CON=0;
    CCP1CON=0;
    TRISBbits.TRISB0=1;      //Configurar RB0 y RC2 como entrada, RC0 como salida
    TRISCbits.TRISC0=0;
    TRISCbits.TRISC2=1;
    ei();                    //Habilitar interrupciones globales
    OPTION_REGbits.INTEDG=0;       //Interrupción externa como flanco de bajada
    PIE1bits.TMR1IE=1;             //Habilitar las interrupciones de CCP1 y
Timer1
    PIE1bits.CCP1IE=1;
    PIR1=0;                  //Poner a cero los flags de interrupción
    while(1);
}

void putch(unsigned char byte)
{
    TXSTA=0x26;
    //RCSTA|=0x80; //
    RCSTAbits.SPEN=1;
    TXREG=byte;
    while(!TXIF)continue;
    TXIF=0;
}

void interrupt ISR(void){

    if(INTCONbits.INTF==1) Pulsador();          //Comprueba si se ha pulsado RB0
    else if (PIR1bits.CCP1IF==1) Flanco();      //Si se ha generado una
interrupción de CCP1
    else if (PIR1bits.TMR1IF==1) Tempor();      //Si hay una interrupción del
                                                //Timer1

}

void Pulsador(void){
    INTCONbits.INTF=0;        //Baja el flag de interrupción
    if (Aux==1) return;       //Comprueba si está realizando una medición
    Trigger();                //Genera el pulso de trigger
    PIR1bits.CCP1IF=0;        //Baja el flag del módulo CCP1
    CCP1CON=0x05;             //Configurar el CCP1 en modo captura
    INTCONbits.PEIE=1;        //Habilita interrupciones de los periféricos
    TMR1L=0;                  //Resetea el Timer1
    TMR1H=0;
```

```
    T1CONbits.TMR1ON=1;        //Inicia el conteo del Timer1
    Aux=1;                     //Activa el indicador de medida en curso
}
void Flanco(void){                   //Calcula la distancia
    PIR1bits.CCP1IF=0;
    if (CCP1CONbits.CCP1M0==1){
         Dist.bytes[0]=CCPR1L;
         Dist.bytes[1]=CCPR1H;
         CCP1CON=0x04;
      }
    else{
         T1CONbits.TMR1ON=0;
         DistTemp.bytes[0]=CCPR1L;
         DistTemp.bytes[1]=CCPR1H;
         Dist.entero=DistTemp.entero-Dist.entero;
         CCP1CON=0;
         INTCONbits.PEIE=0;
         Distancia=(float)Dist.entero*340.0/2.0*0.000001;
         printf("Distancia=%5.3f m\n",Distancia);
         __delay_ms(50);
         Aux=0;
       }
 }

 void Tempor(void){
         PIR1bits.TMR1IF=0;          //Baja el flag
         T1CON=0;                    //Para el Timer1
         CCP1CON=0;                  //Resetea el CCP1
         Dist.entero=0;
         Aux=0;
      }
 void Trigger(void){
     PORTCbits.RC0=1;                //Pulso de disparo
     __delay_us(10);
     PORTCbits.RC0=0;

 }
```

c) Para la simulación del programa del apartado b) será necesario seleccionar la herramienta *MPLABSIM* del menú *debugger/Select Tool* y ajustar la frecuencia del oscilador a 4MHz (menú *debugger / settings*).

La generación de las señales de entrada se realizará mediante el menú *debugger / Stimulus / New Workbook* donde se creará una señal asíncrona que simulará el pulsador conectado a RB0. Para la simulación del pulso de medida se utilizarán las funciones que aparecen en la pestaña *Advanced Pin/Register*. En este caso se crearán dos condiciones que activarán el inicio y el fin del pulso de medida por RC2. En ambos casos la condición será que se active el pulso de trigger por RC0, como se puede observar en la siguiente figura:

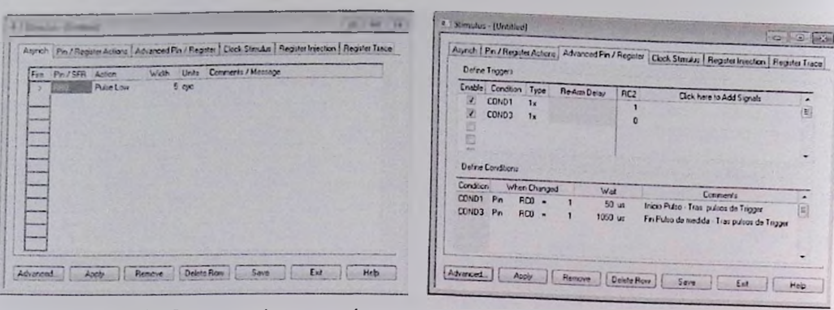


FIGURA 4.11: CONFIGURACIÓN DE LOS ESTÍMULOS EXTERNOS PARA LA SIMULACIÓN DEL EJERCICIO 4.1.3.

Si se ajusta un pulso de entrada por RC2 de duración 1 ms a la entrada de RC2 se obtiene que tras activar el pulsador RB0 se ha guardado en la variable Dist el valor 0x0011 (17 en decimal) que es la distancia en cm a la que se encuentra el objeto y que coincide con el valor obtenido con la fórmula.

La visualización de la señal generada en los terminales RB0, RC0 y RC2 se realizará accediendo al menú *view/Simulator Logic Analyzer* donde pulsando en el botón *channels* se agregarán los canales correspondientes a los pines RB0, RC0 y RC2.

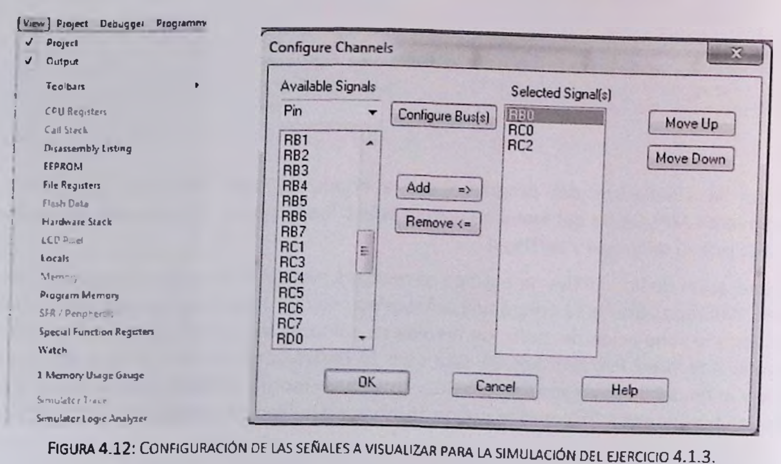


FIGURA 4.12: CONFIGURACIÓN DE LAS SEÑALES A VISUALIZAR PARA LA SIMULACIÓN DEL EJERCICIO 4.1.3.

Una vez ejecutado el programa y activada la condición de disparo (pulsar RB0) se puede observar la señal en los terminales RB0, RC0 y RC2.

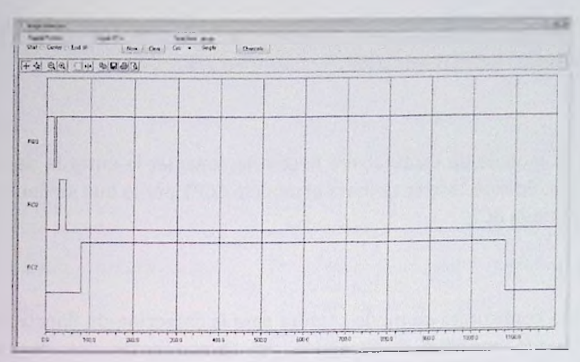


FIGURA 4.13: SIMULACIÓN DEL EJERCICIO 4.1.3 (LOGIC ANALYZER).

d) La duración del pulso viene determinada por los incrementos del TIMER1, que está configurado para incrementarse cada microsegundo. Por lo tanto, el sistema no detectará variaciones en la anchura del pulso menores a 1 microsegundo. Así pues, aplicando la fórmula para obtener la distancia se obtiene que para 1 microsegundo se han recorrido 0.068 cm por lo que en este caso sí se satisface la resolución de 0.3 cm del dispositivo HC-SR04. Sin embargo, la representación del resultado en cm limita la resolución real del sistema a ±1 cm.

4.1.4. Calculo de la velocidad de un motor

Se desea realizar un programa que permita medir la velocidad de un motor en revoluciones por segundo utilizando un encoder de 16 rejillas fijado al eje del motor y un fotodiodo conectado a RC2 que detecta el paso de luz ('1') a través de cada una de las rejillas (Véase figura inferior).

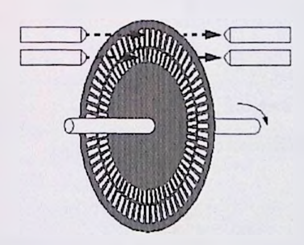


FIGURA 4.14: SISTEMA DE DETECCIÓN ÓPTICA BASADO EN ENCODER DE REJILLAS.

Se pide:

Realiza el programa utilizando el módulo de captura del microcontrolador PIC16F877A trabajando a 4MHz.

Simula el funcionamiento del programa utilizando MPLAB_SIM.

¿Cuáles serán las velocidades máxima y mínima que se podrán detectar utilizando el sistema propuesto?

Solución:

a) Para utilizar el módulo de captura será necesario conectar la entrada del módulo CCPx (RCx) al fotodiodo. En este caso se utilizará el módulo CCP1 por lo que se deberá conectar el fotodiodo a la entrada RC2.

Configuración

El módulo CCP se configurará en modo captura para la detección de flancos de subida con predivisor 1 lo que permitirá la detección del tiempo entre dos flancos de subida consecutivos (tiempo de duración de rejilla).

El Timer 1 asociado al modo captura del módulo CCP se configura como temporizador con predivisor 1 por lo que se incrementará cada 4 ciclos de reloj (1 ciclo de instrucción), es decir, cada microsegundo.

Como se necesita calcular la velocidad en rps se utilizará la siguiente fórmula.

$$rps = \frac{10^6\, {}^{\mu s}/_{s}}{T_{TMR1}\, {}^{\mu s}/_{rejilla} \cdot 16\, {}^{rejillas}/_{vuelta}} = \frac{62500}{T_{TMR1}}\, {}^{vueltas}/_{s}$$

Código fuente

```
#include <htc.h>
#include <stdio.h>
#include "timers.h"
#include "ccp.h"

__CONFIG(FOSC_HS & WDTE_OFF & LVP_OFF & PWRTE_ON);
#define _XTAL_FREQ 4000000

typedef union {
    unsigned char bytes[2];
    unsigned int entero;} ubyte2;

    ubyte2 Vel, Div, T,Ttemp;

float Velocidad;
void Captura(void);

void main(void){
    di();                           //Deshabilita interrupciones
    Captura();                      //Calcula la velocidad
    while(1);
}

void putch(unsigned char byte){     //Función para permitir el uso de printf con
                                    //el puerto serie
    TXSTA=0x26;
    RCSTAbits.SPEN=1;
```

```
    TXREG=byte;
    while(!TXIF)continue;
    TXIF=0;
}

void Captura(void){
    setup_ccp1(CCP_OFF);                        //Reset al módulo CCP1
    setup_timer1(T1_INT|T1_DIV1|T1_OFF);        //Timer1 como temporizador,
                                                //pred=1 y detenido.
    set_timer1(0x00);
    PIR1bits.CCP1IF=0;                    //Poner a 0 el indicador (flag) de
captura
    setup_ccp1(CCP_CAPTURE_RE);;          //Seleccionar modo captura con 1 flanco
                                          //de subida
    start_timer1();                       //Arrancar el TIMER1
    while(PIR1bits.CCP1IF==0);            //Comprobar si ha llegado un nuevo
                                          //flanco de subida
    T.entero=CCPR1;                       //Recupera la primera captura
    PIR1bits.CCP1IF=0;
    while(PIR1bits.CCP1IF==0);            //Comprobar si ha llegado un nuevo
                                          //flanco de subida
    Ttemp.entero=CCPR1;                   //Recupera la segunda captura
    T.entero=Ttemp.entero-T.entero;
    setup_ccp1(CCP_OFF);                  //Reset al módulo CCP1
    stop_timer1();                        //Para el Timer 1
    Vel.entero=62500/T.entero;            //Velocidad en rps
    Velocidad=62500.0/T.entero;
    printf("Velocidad=%d rps (%5.3f)",Vel.entero,Velocidad);
}
```

b) Para la simulación del programa del apartado b) será necesario seleccionar la herramienta *MPLABSIM* del menú *Debugger/Select Tool* y ajustar la frecuencia del oscilador a 4MHz (menú *debugger / settings*).

La generación de las señales de entrada se realizará mediante el menú *debugger / Stimulus / New Workbook* donde se creará una señal periódica de entrada por RC2. En este caso se elegirá una señal periódica de 1 ms por lo que aplicando la fórmula vista en el apartado a) se obtiene que dicha señal representa una velocidad de 62.5 rps.

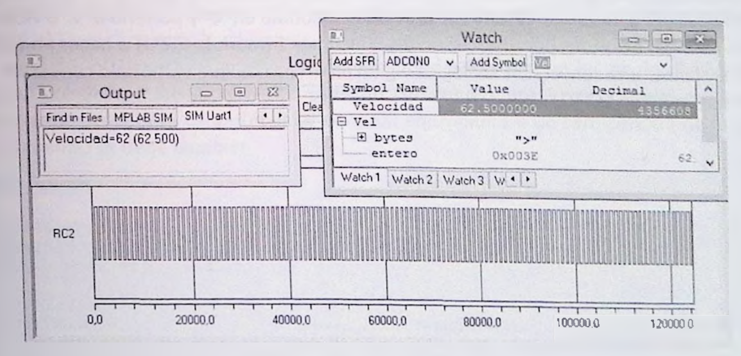


FIGURA 4.15: SIMULACIÓN DEL EJERCICIO 4.1.4.

Si se ejecuta el programa con la señal de entrada por RC2 se observa el resultado almacenado en la variable *Velocidad* se corresponde con la velocidad en rps dada por la fórmula del apartado a).

c) A la vista del sistema propuesto en el apartado a) se puede deducir que la velocidad máxima y mínima del sistema vendrán determinadas por la fórmula:

$$rps = \frac{62500}{T_{TMR1}} vueltas/_{s}$$

Donde T_{TMR1} es el tiempo almacenado en los registros asociados al Timer 1 en microsegundos. El valor almacenado por el Timer 1 va de 1 a 65535 pero se toma desde 1 a 62500, ya que para el valor 0 el motor iría demasiado rápido para poder ser medido y para valores mayores a 62500 el sistema iría por debajo de 1 rps. Así pues, el rango de velocidades de medida del sistema estaría entre 62500 y 1 rps.

Siendo estrictos, la velocidad máxima que se puede obtener con dicho programa no alcanzaría las 62500 rps ya que se necesitan ejecutar una serie de instrucciones entre una captura y la siguiente (aproximadamente 8 ciclos de instrucción, 8 µs, lo que haría que la velocidad máxima real fuese de 7812 rps).

Modifica el programa del apartado a) para que funcione mediante interrupciones.

4.2. Módulo Comparación

El funcionamiento de los módulos CCPx en modo comparación permite al microcontrolador PIC16F877A generar un evento cuando el valor del contador de 16 bits asociado al Timer 1 iguala al valor almacenado en los registros CCPRxH:CCPRxL del módulo CCPx. Los posibles eventos podrán estar o no relacionados con el terminal de salida asociado al módulo. Estos eventos permitirán desde únicamente activar el flag indicador de que se ha producido el evento (CCPxIF) a inicializar el terminal asociado al módulo en '0' y ponerlo a '1' o viceversa o en el caso del evento especial hacer un reset al Timer 1 (módulo CCP1) o hacer un reset al Timer 1 y lanzar una conversión analógica digital en el caso de que el módulo conversor esté habilitado (módulo CCP2).

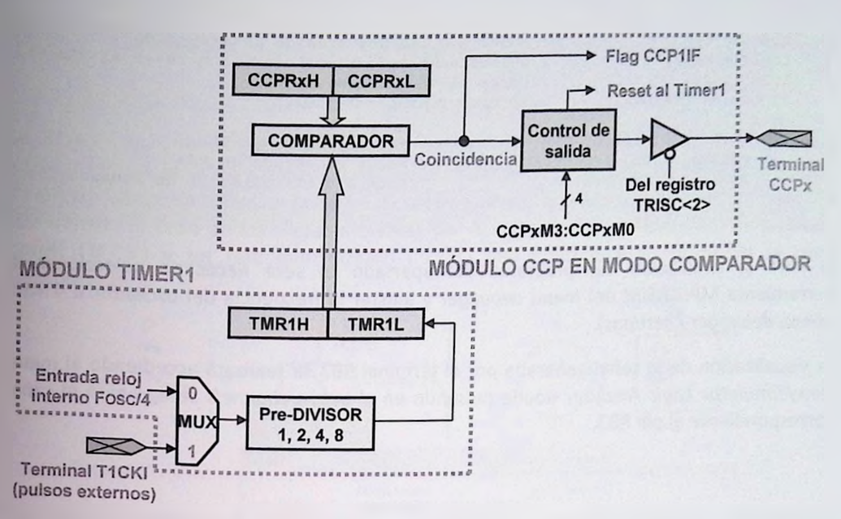


FIGURA 4.16: ESQUEMA DE BLOQUES DEL MÓDULO COMPARACIÓN.

4.2.1. Temporizador automático de 16 bits

Realiza un programa en C que genere una señal de período 10 ms a la salida del puerto RB3 mediante la utilización del modo comparación de un microcontrolador PIC16F877A funcionando a 4 MHz y simular su funcionamiento con *MPLAB_SIM*.

Solución:

Para la realización del programa es necesario configurar el Timer1 como temporizador de 16 bits empleando los registros CCPR1H y CCPR1L para almacenar el módulo de conteo. Para ello se programa el módulo CCP1 en modo comparador, usando la variante de generar un reset al Timer1 cuando la comparación entre los registros CCPR1 y TMR1 sea positiva. En este ejemplo, el Timer1 se programa como temporizador, con un factor de división de 1 para su pre-divisor (se incrementa cada microsegundo) por lo que en el módulo de comparación se deberá cargar el valor de 5000 equivalente a un semiperiodo de la señal, que es cuando se debe cambiar.

Código fuente

```
#include <htc.h>
__CONFIG(FOSC_HS & WDTE_OFF & LVP_OFF & PWRTE_ON);
#define _XTAL_FREQ 4000000

void main(void){
    T1CON=0;                //Timer1 como temporizador, pred=1 (1µs), detenido.
    CCP1CON=0;              //Reset al módulo CCP1.
    INTCON=0;               //Deshabilita interrupciones.
    TRISBbits.TRISB3=0;     //Configurar el terminal RB3 como salida.
```

```
    PIR1=0;                       //Poner a 0 las banderas de interrupción.
    CCP1CON=0x0B;                 //Seleccionar modo comparador, con reset al Timer1.
    CCPR1=5000;                   //Carga el módulo comparador
    T1CONbits.TMR1ON=1;           //Iniciar conteo del Timer1.
    while(1){
        while(PIR1bits.CCP1IF==0) continue;    //¿CCPR1 = TMR1?
        PORTBbits.RB3=!PORTBbits.RB3;     //Negar el estado de RB3
        PIR1bits.CCP1IF=0;                //Pone a 0 el indicador de comparación.
    }
}
```

c) Para la simulación del programa del apartado b) será necesario seleccionar la herramienta *MPLABSIM* del menú *debugger* y ajustar la frecuencia del oscilador a 4 MHz (menú *debugger / settings*).

La visualización de la señal generada por el terminal RB3 se realizará accediendo al menú *view/Simulator Logic Analyzer* donde pulsando en el botón *channels* se agregará el canal correspondiente al pin RB3.

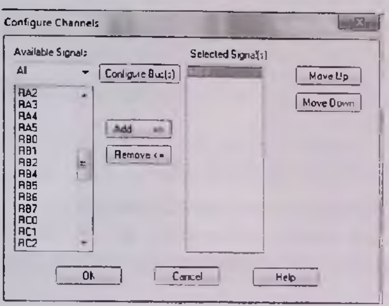


FIGURA 4.17: CONFIGURACIÓN DE LAS SEÑALES A VISUALIZAR PARA LA SIMULACIÓN DEL EJERCICIO 4.2.1.

Una vez ejecutado el programa se puede observar la señal a la salida del terminal RB3 y obtener su período aproximado utilizando los cursores.

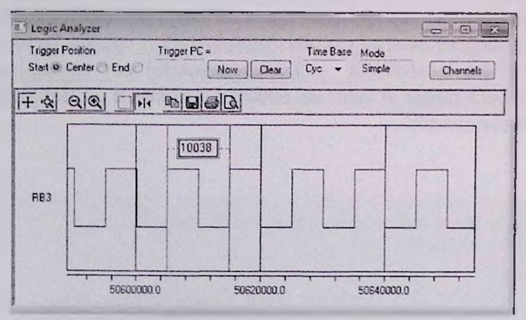


FIGURA 4.18: SIMULACIÓN DEL EJERCICIO 4.2.1.

Modifica el programa anterior para que funcione mediante interrupciones.

4.2.2. Contador de cajas en cinta transportadora

Se desea implementar un sistema que cuente las piezas que pasan por una cinta transportadora. El sistema deberá parar la cinta cuando haya contado 10 piezas y arrancará de nuevo cuando se active un pulsador. El sistema estará formado por un detector inductivo que detecta el paso de las piezas metálicas por la cinta transportadora ('1'), un pulsador on/off ('0'/'1') y un relé marcha/paro ('1'/'0') que controlará el motor de la cinta transportadora.

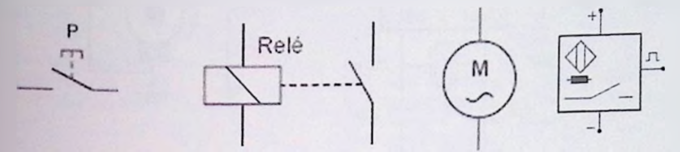


FIGURA 4.19: REPRESENTACIÓN ESQUEMÁTICA DE LOS ELEMENTOS QUE SE UTILIZARÁN EN EL EJERCICIO 4.2.2

Se desea implementar el sistema utilizando un microcontrolador PIC16F877A (Fosc = 4 MHz) utilizando el modo comparación.

Se pide:

a) Indica las conexiones de los diferentes elementos con el microcontrolador.

b) Realiza el programa en C que controle el sistema.

c) Simula el funcionamiento del programa con *MPLAB_SIM*.

Solución:

a) Para el funcionamiento del microcontrolador es necesario conectar correctamente las entradas V_{DD} y V_{SS} a +5 V y 0 V respectivamente. Además, como el PIC16F877A no cuenta con oscilador interno, será necesario conectar el oscilador (en este caso un oscilador de cristal a 4 MHz) a las entradas OSC1 y OSC2 tal y como aparece en la figura 4.20.

El pulsador se conectará a una entrada digital del microcontrolador. En este caso se ha utilizado la entrada RB0, que permitirá el funcionamiento mediante interrupción.

La utilización del módulo comparación implica que el valor almacenado en los registros TMR1H:TMR1L será comparado con el valor almacenado en los registros CCPxH:CCPxL. En este caso se utilizará el TIMER1 como contador por lo que la entrada del sensor se deberá conectar al terminal RC0/T1CKI. Además, el módulo comparador permite ser configurado de forma que cuando el valor almacenado en el TIMER1 iguale al valor almacenado por el módulo comparador se modifique la salida asociada a dicho módulo. Por lo tanto, si se utiliza el módulo CCP1 se conectará la salida RC2 al relé que controla el motor (en el caso de utilizar el módulo CCP2 se debería conectar el relé a la salida RC1).

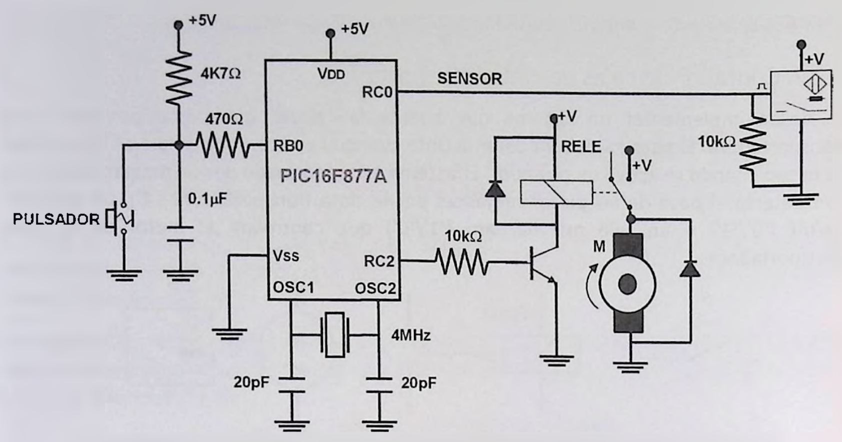


FIGURA 4.20: ESQUEMA DE CONEXIONES CON EL MICROCONTROLADOR DEL SISTEMA CONTADOR DE CAJAS EN CINTA TRANSPORTADORA DEL EJERCICIO 4.2.2.

b) Para contar las piezas que pasan por la cinta transportadora se utilizará el TIMER1 configurado en modo contador de pulsos (flancos de subida) por el terminal RC0/T1CKI que se incrementará cada vez que se detecte una pieza y que combinado con el módulo comparador permitirá evaluar si se ha llegado al número de piezas deseado.

Configuración

El terminal RB0 se configurará para funcionar mediante interrupción por flanco de subida (por defecto).

El TIMER1 se configurará en modo contador por RC0 con predivisor 1 (un incremento por cada pieza).

El módulo CCP se configurará en modo comparador de forma que la salida RC2, que controla el motor, pase a estado bajo ('0') cuando se alcance el valor 10 almacenado en los registros del comparador.

Código guente

```
#include <htc.h>
#include <stdio.h>
#include "interrupts.h"
#include "ports.h"
#include "timers.h"
#include "ccp.h"

__CONFIG(FOSC_HS & WDTE_OFF & LVP_OFF & PWRTE_ON);
#define _XTAL_FREQ 4000000
#define Piezas 10;

void Pulsador(void);
void Compara(void);
```

```
void main(void){
    enable_int(GLOBAL|INT_EXT|INT_CCP1);      //Habilita interrupciones
    bit_clear(TRISC,2);                       //Configurar el terminal RC2 como
                                              //salida.
    OPTION_REGbits.INTEDG=0;                  //Interrupción externa por flanco
                                              //de bajada.
    while(1);
    }

void interrupt ISR(void){
    if(INTCONbits.INTF==1&&INTCONbits.INTE) Pulsador();   //Comprueba si se ha
                                                          //pulsado RB0
    else if (PIR1bits.CCP1IF==1&&PIE1bits.CCP1IE) Compara();  //Comprueba
                                                              //interrupción CCP1
}

void Pulsador(void){
    INTCONbits.INTF=0;                      //Baja el flag de interrupción por RB0
    if(PORTCbits.RC2==1) return;            //Comprueba si el motor está parado.
    PIR1=0;                                 //Poner a 0 las banderas de
                                            //interrupción.
    enable_int(PERIPHERAL);;                //Habilita interrupciones de periféricos
    setup_timer1(T1_EXT|T1_DIV1|T1_OFF);    //Timer1 contador de pulsos, pred=1,
                                            //detenido.
    set_timer1(0x00);                       //Pone a 0 los contadores del TIMER1
    set_ccp1(Piezas);                       //Carga los valores para el módulo
                                            /7comparador
    setup_ccp1(CCP_COMPARE_FALL);           //Seleccionar modo comparador, con
                                            //puesta a '0'de RC2
    start_timer1();                         //Inicia el conteo del Timer1.
}
void Compara(void){
    PIR1bits.CCP1IF=0;                      //Baja el flag del módulo CCP1
    setup_ccp1(CCP_OFF);                    //Reset al módulo CCP1
    stop_timer1();                          //Para el Timer1.
    disable_int(PERIPHERAL);                //Deshabilita las interrupciones de
                                            //periféricos.
}
```

c) Para la simulación del programa del apartado b) será necesario seleccionar la herramienta *MPLABSIM* del menú *debugger* y ajustar la frecuencia del oscilador a 4MHz (menú *debugger / settings*).

La simulación del pulsador se realizará mediante el menú *debugger / Stimulus / New Workbook* donde se creará un estímulo asíncronos asociados a la entrada RB0 (pulsador) y un estímulo periódico asociado al fotodetector RC0 (fotodetector). Para facilitar la simulación se seleccionará una señal de frecuencia elevada en RC2 (por ejemplo 100 KHz).

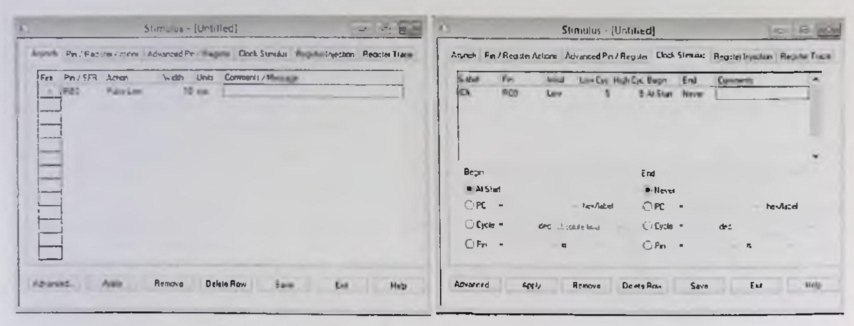


FIGURA 4.21: CONFIGURACIÓN DE LOS ESTÍMULOS EXTERNOS PARA LA SIMULACIÓN DEL EJERCICIO 4.2.2.

La visualización de las señales generadas se realizará accediendo al menú *view/Simulator Logic Analyzer* donde pulsando en el botón *channels* se agregarán los canales correspondientes a los terminales RB0 (pulsador), RC0 (fotodetector) y RC2 (relé).

Se ejecutará el programa en modo *animate* y se irán activando los estímulos (*fire*) comenzando por RB0 y siguiendo por RC2. Por simplicidad, se ha simulado en la figura inferior un contador de 15 piezas.

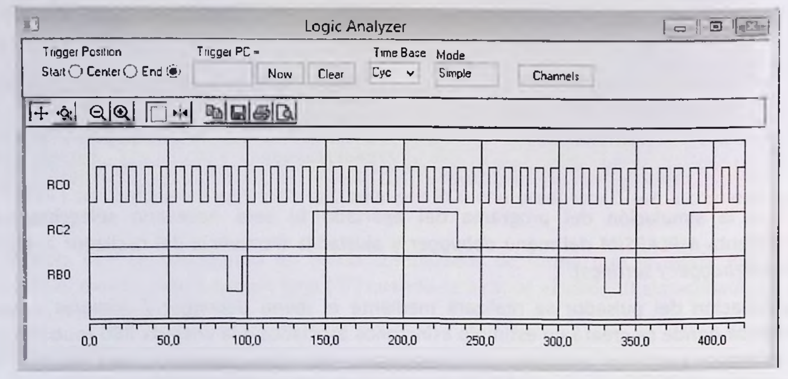


FIGURA 4.22: SIMULACIÓN DEL EJERCICIO 4.2.2.

4.3. Módulo PWM

Cuando el módulo CCPx funciona en modo PWM permite generar a través de su terminal asociado una señal cuadrada periódica con ciclo de trabajo variable con una resolución máxima de 10 bits en el mejor de los casos. El período de la señal vendrá fijado por la

temporización del Timer 2 (excluyendo el postdivisor) mientras que el ciclo de trabajo se fijará mediante el registro CCPRxL.

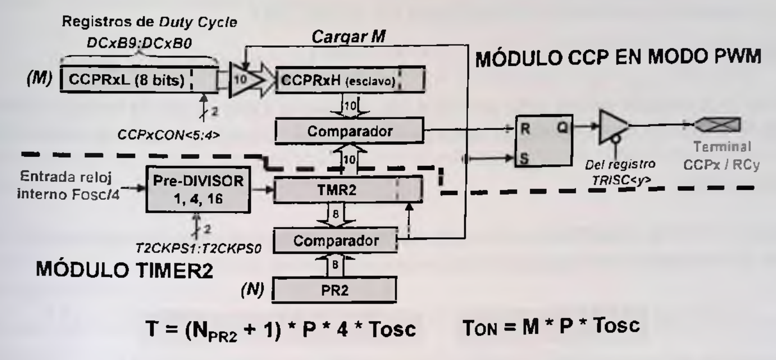


FIGURA 4.23: ESQUEMA DE BLOQUES DEL MÓDULO PWM.

4.3.1. Generación de tonos

Se desea generar un tono de frecuencia 1KHz y ciclo de trabajo 50 % a través del zumbador conectado al terminal RC2 de un microcontrolador PIC16f877A trabajando a 4 MHz (véase Figura). Para ello se utilizará el módulo CCP1 configurado en modo PWM.

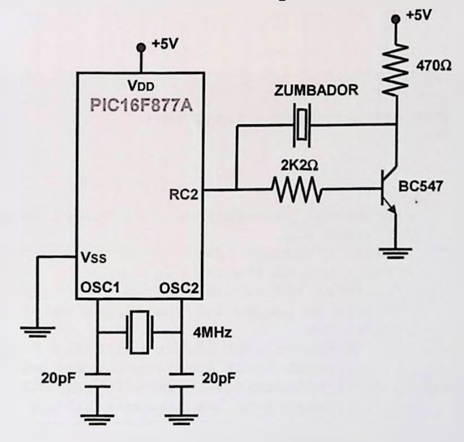


FIGURA 4.24: CONEXIONES CON EL MICROCONTROLADOR PARA LA GENERACIÓN DE UN TONO DE 1 KHZ DEL EJERCICIO 4.3.1.

Se pide:

a) Implementa el código en ensamblador que cumpla las especificaciones descritas en el enunciado.

b) Indica la frecuencia máxima y mínima de las señales PWM que pueden ser generadas con dicho sistema.

c) Simula el funcionamiento del programa con *MPLAB_SIM*

Solución:

a) Para la generación de una señal periódica con frecuencia 1 KHz y ciclo de trabajo 50 % se utilizará el módulo CCP1 asociado al Timer 2 cuya salida se corresponde con el terminal RC2.

Configuración

El período T de la señal PWM se fija mediante el valor N depositado en el registro PR2. Así pues, para una señal de frecuencia 1 KHz tenemos un período T = 1 ms.

$$T = (N + 1) * P * 4 * T_{osc} \Rightarrow 1\ ms = (N + 1) * P * 4 * 0{,}25\ \mu s \Rightarrow 1000 = (N + 1) * P$$

Si se escoge el factor de división del pre-divisor del Timer2 P=4, el valor que hay que colocar en el registro PR2 es N = 249.

Utilizando solo el registro CCPR1L (M_8) para fijar el valor de T_{ON} de los pulsos. Los bits CCP1CON<5:4> (M_2) se mantendrán permanentemente en 0. Entonces:

$$T_{ON} = 4 * M_8 * P * T_{osc} \Rightarrow 50\ \% * 1\ ms = 4 * M_8 * 4 * 0{,}2\ \mu s \Rightarrow M8 = 125$$

Código fuente

```
#include <htc.h>

__CONFIG(FOSC_HS & WDTE_OFF & LVP_OFF & PWRTE_ON);
#define _XTAL_FREQ 4000000

void main(void){

    T2CON=0x01;           //Programar post-divisor = 1, Timer2 detenido y pre-
                          //divisor = 4.
    CCP1CON=0;            //Reset al módulo CCP1
    TMR2=0;               //Poner a 0 el Timer2
    CCPR1L=125;           //La señal PWM saldrá con 50%
    PR2=249;              //Módulo de conteo del Timer2 que es el período de la
                          //señal PWM.
    PIE1bits.TMR2IE=0;         //Inhabilitar interrupción del Timer2
    PIE1bits.CCP1IE=0;         //Inhabilitar interrupción del módulo CCP1.
    TRISCbits.TRISC2=0;        //Configurar el terminal CCP1/RC2 como salida.
    PIR1=0;                    //Desactivar las banderas (flags)de interrupción.
    CCP1CON=0x0C;              //El módulo CCP1 en modo PWM con los bits
                               //DC1B1:DC1B0 en 0 (M2=0).
    T2CONbits.TMR2ON=1;        //Iniciar conteo del Timer2.
    while(1);
}
```

b) Las frecuencias máximas y mínimas de la señal PWM vendrán determinadas por los valores de configuración de la temporización del Timer 2 sin tener en cuenta el postdivisor. Así pues, tomando los valores máximos y mínimos para cada caso se obtiene:

$$T_{PWM} = (\mathrm{PR2} + 1) * \mathrm{P} * 4 * T_{osc} \Rightarrow \begin{cases} T_{max} = (255 + 1) * 16 * 1\mu s = 4096\mu s;\ f_{min} \sim 244.1Hz \\ T_{min} = (1 + 1) * 1 * 1\mu s = 2\mu s;\ f_{max} = 0.5MHz \end{cases}$$

Es importante destacar que en el cálculo del período mínimo no se puede introducir un valor de PR2 = 0 ya que siempre se cumpliría que TMR2 = PR2 y no avanzaría el contador TMR2.

c) Para observar la señal generada por el terminal RC2 es necesario acceder al menú *view/Simulator Logic Analyzer* donde pulsando en el botón *channels* se agregará el canal correspondiente al pin RC2.

Una vez ejecutado el programa o ejecutándolo en modo *animate* se puede observar la señal a la salida del terminal RC2 y comprobar su período y ciclo de trabajo.

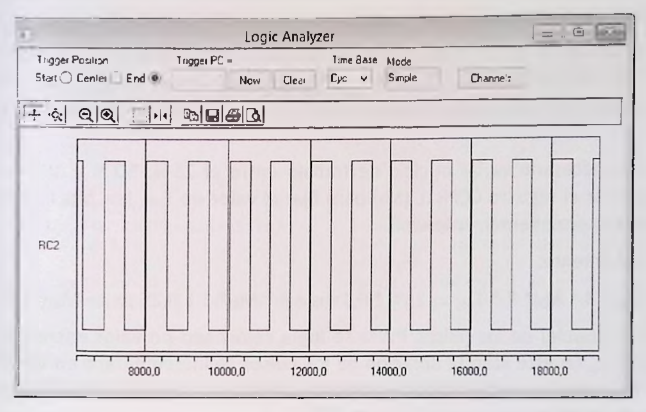


FIGURA 4.25: SIMULACIÓN DEL EJERCICIO 4.3.1.

4.3.2. Señal periódica con ciclo de trabajo variable

Se desea generar una señal PWM con un período constante de 0,1 ms y ciclo de trabajo variable (inicialmente con valor 25 %). El ciclo de trabajo se doblará por cada pulsación de RB0 hasta llegar al 100 % volviendo al 25 % en la siguiente pulsación.

a) Implementa el código en ensamblador para un microcontrolador PIC16F877A trabajando a 4 MHz que cumpla las especificaciones del problema.

b) Simula el funcionamiento del programa utilizando *MPLAB_SIM*.

c) Indica la resolución (variación mínima del ciclo de trabajo de la señal PWM) del sistema propuesto

Solución:

a) Para la generación de una señal periódica con ciclo de trabajo variable se utilizará el módulo CCPx del microcontrolador asociado al Timer 2. Para la modificación del ciclo de trabajo se utilizará la interrupción externa conectando el pulsador al terminal RB0 del microcontrolador.

Configuración

El periodo T de la señal PWM se fija mediante el valor N depositado en el registro PR2.

$$T = (N + 1) * P * 4 * T_{osc} \Rightarrow 0{,}1 \text{ ms} = (N + 1) * P * 4 * 0{,}25\ \mu s \Rightarrow 100 = (N + 1) * P$$

donde P es el factor de división del pre-divisor del Timer2, P = 1, 4, 16.

Si se escoge P = 1, el valor que hay que colocar en el registro PR2 será N = 99.

La duración TON de los pulsos queda determinada por el valor M de los 10 bits distribuidos entre los 8 bits del registro CCPR1L (que constituyen los 8 bits más significativos de M, es decir, M_8) y los 2 bits CCP1CON<5:4>, que forman los 2 bits menos significativos de M, es decir, M_2.

Como solo es necesario variar el ciclo de trabajo entre el 25 %, 50 % y 100 % solo será necesario utilizar el registro CCPR1L (M_8) para fijar el valor de T_{ON}. Los bits CCP1CON<5:4> se mantendrán permanentemente en 0.

Entonces inicialmente:

$$T_{ON} = 4 * M_8 * P * T_{osc} \Rightarrow 25\% * 0{,}1 \text{ ms} = 4 * M_8 * 1 * 0{,}25\ \mu s \Rightarrow M_8 = 25$$

La duración (variable) de los pulsos PWM se logra colocando un valor entre 0 y 99 en el registro CCPR1L. En este caso la duración de los pulsos se incrementará en un factor de 2 por lo que bastará con rotar el valor del registro CCPR1L una posición hacia la izquierda por cada pulsación hasta que alcance el valor de 100.

Código fuente

```
#include <htc.h>
#include <stdio.h>
#include "timers.h"
#include "ccp.h"

__CONFIG(FOSC_HS & WDTE_OFF & LVP_OFF & PWRTE_ON);
#define _XTAL_FREQ 4000000

typedef union {
    unsigned char bytes[2];
    unsigned int entero;} ubyte2;
```

```
ubyte2 DutCy;

void Inic_pwn(void);

void main(void){
    DutCy.bytes[0]=25;
    Inic_pwn();  //programar el módulo CCP1 en modo PWM para generar
                 //una señal PWM de periodo 0,1 ms y ciclo de trabajo del 25%.
    INTCON=0x90;       //Habilita interrupciones globales e interrupción externa
    while(1);
}

void putch(unsigned char byte){
    TXSTA=0x26;
    RCSTAbits.SPEN=1;
    TXREG=byte;
    while(!TXIF)continue;
    TXIF=0;
}

void Inic_pwn(void){
    setup_timer2(T2_POST_DIV1|T2_OFF|T2_PRED_DIV1); //Timer2 detenido,
                                                    //PRED=POSTD=1
    setup_ccp1(CCP_OFF);                 //Reset al módulo CCP1.
    set_pwm1_duty(DutCy.entero<<2);      //Señal PWM saldrá con ciclo de trabajo
                                         //25 %.
    set_timer2(99);            //Módulo de conteo del Timer2 que es el periodo de
                               //la señal PWM
    TRISCbits.TRISC2=0;                  //Poner el terminal CCP1 como salida.
    setup_ccp1(CCP_PWM);                 //El módulo CCP1 en modo PWM
    start_timer2();                      //Iniciar conteo del Timer2
}

void interrupt ISR(void){
    if (INTCONbits.INTF==0) return;      //Comprueba la interrupción externa
    INTCONbits.INTF=0;                   //Baja el indicador de interrupción
                                         //externa
    DutCy.bytes[0]<<=1;                  //Multiplica por 2 el ciclo de trabajo
    if (DutCy.bytes[0]>100) DutCy.bytes[0]=25;      //Comprueba si el ciclo de
                                                    //trabajo es del 100%
    set_pwm1_duty(DutCy.entero<<2);                 //Ajusta el ciclo de trabajo
    printf("DC=%d\n",CCPR1L);
}
```

b) Para la simulación del programa del apartado a) será necesario seleccionar la herramienta *MPLABSIM* del menú *debugger* y ajustar la frecuencia del oscilador a 4MHz (menú *debugger / settings*).

La simulación del pulsador se realizará mediante el menú *debugger / Stimulus / New Workbook* donde se creará una señal asíncrona de duración 5 ciclos de instrucción asociada a RB0 (*pulse low)*.

La visualización de la señal generada por el terminal RC2 se realizará accediendo al menú *view/Simulator Logic Analyzer* donde pulsando en el botón *channels* se agregará el canal correspondiente al pin RC2.

Una vez ejecutado el programa se puede observar la señal a la salida del terminal RC2 y comprobar su ciclo de trabajo tras cada pulsación de RB0 (*fire)*.

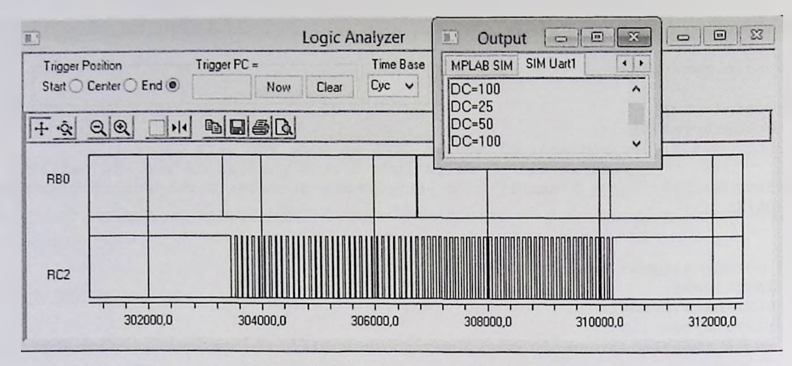


FIGURA 4.26: SIMULACIÓN DEL EJERCICIO 4.3.2.

c) La resolución del sistema propuesto en el apartado a) dependerá del valor asignado a PR2. En este caso PR2 = 99 por lo que el ciclo de trabajo de la señal PWM se podrá ir incrementando de 0 % a 100 % en incrementos de 0.25 % en el caso de utilizar los bits CCP1CON<5:4> o en incrementos del 1 % en el caso de no utilizarlos. Así pues, el incremento mínimo del ciclo de trabajo será:

$$Utilizando\ CCP1CON < 5:4 > \ \Rightarrow Resolución\ (ms) = \frac{T}{(PR2+1)*4} = \frac{0.1}{(99+1)*4} = 0.00025ms$$

$$Sin\ utilizar\ CCP1CON < 5:4 > \ \Rightarrow Resolución\ (ms) = \frac{T}{(PR2+1)} = \frac{0.1}{(99+1)} = 0.001ms$$

4.3.3. Control motor DC

Se desea controlar la velocidad de un motor DC mediante la variación del ciclo de trabajo de la señal de alimentación.

Para ello se utilizará el módulo CCP1 configurado en modo PWM de un microcontrolador PIC16F877A trabajando a 4 MHz (Tosc = 0,25 µs).

Para evitar alimentar el motor con un rizado de corriente alto, y por lo tanto de par, la señal PWM tendrá un periodo superior a 20KHz. Con esto se conseguirá que el motor gire a una velocidad más estable.

Además, el sistema contará con un pulsador que variará la tensión media aplicada al motor entre 0 %, 25 %, 50 %, 75 %, 100 %, 0 %, 25 %, etc., por cada pulsación detectada.



FIGURA 4.27: MOTOR DE CORRIENTE CONTINUA.

a) Indica las conexiones del sistema microcontrolador.

b) Implementa el código del programa en ensamblador que cumpla las especificaciones del problema.

c) Simula el funcionamiento del programa utilizando *MPLAB_SIM*.

Solución:

a) Para el funcionamiento del microcontrolador es necesario conectar correctamente las entradas V_{DD} y V_{SS} a +5 V y 0 V respectivamente. Además, será necesario conectar el oscilador, en este caso un oscilador de cristal a 4MHz a las entradas OSC1 y OSC2 tal y como aparece en la figura inferior.

El interruptor se conectará a una entrada digital del microcontrolador. En este caso se ha utilizado la entrada RB0, que se corresponde con la entrada de interrupción externa del microcontrolador, lo que permitirá al programa funcionar en modo interrupción.

La utilización del módulo CCP1 implica que la señal PWM saldrá por el terminal RC2 que deberá conectarse al motor. Sin embargo, la corriente demandada por un motor supera generalmente el máximo de corriente suministrada por el terminal (20 mA) por lo que es necesaria la utilización de una etapa de salida (en este caso se utiliza un buffer con entrada *ENABLE* por RC0). Además, se ha añadido un diodo de protección para evitar sobretensiones en el driver cuando se corta la corriente. Las conexiones con el motor y el micro pueden variar en función del driver utilizado.

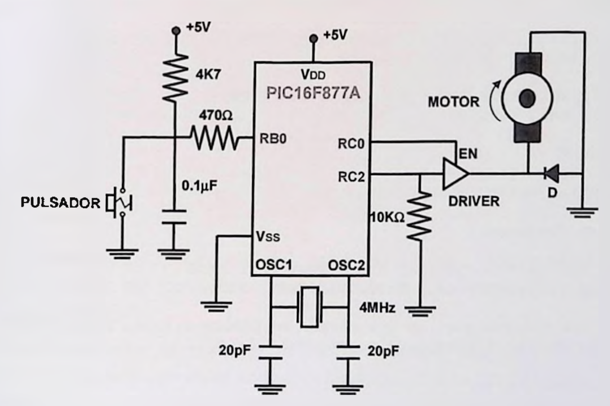


FIGURA 4.28: CONEXIONES DEL CONTROLADOR PARA EL SISTEMA DE CONTROL DE VELOCIDAD DE UN MOTOR DC DEL EJERCICIO 4.3.3.

b) Para la generación de una señal periódica con ciclo de trabajo variable se utilizará el módulo CCPx del microcontrolador asociado al Timer 2. Para la modificación del ciclo de

trabajo se utilizará la interrupción externa conectando el pulsador al terminal RB0 del microcontrolador.

Configuración

El período T de la señal PWM se fija mediante el valor N depositado en el registro PR2. Así, para una frecuencia superior a 20 kHz se tendrá un período T < 50 µs.

$$T = (N + 1) * P * 4 * T_{osc} \Rightarrow 50\ \mu s = (N + 1) * P * 4 * 0{,}25\ \mu s \Rightarrow 100 = (N + 1) * P$$

donde P es el factor de división del pre-divisor del Timer2, P = 1, 4, 16.

Si se escoge P = 1, el valor que hay que colocar en el registro PR2 será N = 49.

La duración TON de los pulsos queda determinada por el valor M de los 10 bits distribuidos entre los 8 bits del registro CCPR1L (que constituyen los 8 bits más significativos de M, es decir, M_8) y los 2 bits CCP1CON<5:4>, que forman los 2 bits menos significativos de M, es decir, M_2.

Como sólo es necesario variar el ciclo de trabajo entre el 25 %, 50 %, 75 % y 100 % se utilizará una tabla donde se almacenarán los valores que se cargarán en el registro CCPR1L (M_8) y en los bits CCP1CON<5:4> para ajustar el valor de T_{ON}.

Los valores para cada ciclo de trabajo se obtienen utilizando la siguiente fórmula:

$$T_{ON} = DC * T = (M_8:M_2) * P * T_{osc}$$

Código fuente

```
#include <htc.h>
#include <stdio.h>

__CONFIG(FOSC_HS & WDTE_OFF & LVP_OFF & PWRTE_ON);
#define _XTAL_FREQ 4000000

typedef union {
    unsigned char bytes[2];
    unsigned int entero;} ubyte2;

unsigned char Contador;

//Tabla con los valores que hay que cargar en CCPR1L
unsigned char Tabla1[5]={0x32,0x25,0x19,0x0C,0x00};

//Tabla con los valores que hay que cargar en CCPCON<5:4> en la posición adecuada
unsigned char Tabla2[5]={0x00,0x20,0x00,0x20,0x00};

void Inic_pwn(void);
void Ton_pwm(void);

int main()
{
    Contador=5;          //Inicia el contador con el número de valores que puede
                         //tomar el DC.
    Inic_pwn();          //programar el módulo CCP1 en modo PWM para generar PWM
                         //de 20 kHz, DC=0
    INTCON=0x90;         //Habilita interrupciones globales e interrupción externa
```

```
    OPTION_REGbits.INTEDG=0;        //Interrupción por flanco de bajada
    while(1);
}

void putch(unsigned char byte)
{
    TXSTA=0x26;
    RCSTAbits.SPEN=1;
    TXREG=byte;
    while(!TXIF)continue;
    TXIF=0;
}
void Inic_pwn(void){
    T2CON=0;                //Programar post-divisor = 1, Timer2 detenido y pre-
                            //divisor = 1
    CCP1CON=0;              //Reset al módulo CCP1.
    TMR2=0;                 //Poner a 0 el Timer2
    CCPR1L=0;               //La señal PWM saldrá con ciclo de trabajo del 0%.
    PR2=49;                 //Módulo de conteo del Timer2 que es el periodo de la
                            //señal PWM.
    TRISCbits.TRISC2=0;           //Poner el terminal CCP1 como salida
    TRISCbits.TRISC0=0;           //Poner el terminal RC0 como salida
    PORTCbits.RC0=1;              //Habilitar la entrada ENABLE del DRIVER
    CCP1CON=0x0C;                 //El módulo CCPI en modo PWM con los bits
                                  //DC1B1:DC1BO en 0 (M2=0).
    T2CONbits.TMR2ON=1;           //Iniciar conteo del Timer2.
}
void interrupt ISR(void){
    if(INTCONbits.INTF==0) return;      //Comprueba si ha saltado la
                                        //interrupción externa
    INTCONbits.INTF=0;                  //Baja el indicador de interrupción
                                        //externa
    if(--Contador==0) Contador=5;       //Decrementa el contador y recarga si
                                        //es cero
    Ton_pwm();                          //Ajusta el nuevo ciclo de trabajo
}
void Ton_pwm(void){
    CCPR1L=Tabla1[Contador-1];          //Ajusta el ciclo de trabajo
    CCPR1H=Tabla1[Contador-1];
    CCP1CON|=Tabla2[Contador-1];
    CCP1CON&=Tabla2[Contador-1]+0x0F;
    printf("CCPR1L=%d,CCP1CON=%d, Contador=%d\n",CCPR1L,CCP1CON,Contador);
}
```

c) Para la simulación del programa del apartado b) será necesario seleccionar la herramienta *MPLABSIM* del menú *debugger* y ajustar la frecuencia del oscilador a 4MHz (menú *debugger / settings*).

La simulación del pulsador se realizará mediante el menú *debugger / Stimulus / New Workbook* donde se creará una señal asíncrona de duración 5 ciclos de instrucción asociada a RB0 (*pulse low*).

La visualización de la señal generada por el terminal RC2 se realizará accediendo al menú *view/Simulator Logic Analyzer* donde pulsando en el botón *channels* se agregará el canal correspondiente al pin RC2.

Se ejecutará el programa en modo *animate* y se observará la señal a la salida del terminal RC2 tras cada pulsación de RB0 (*fire*).

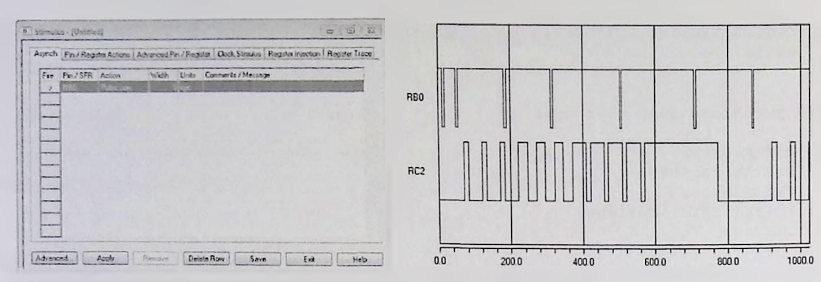


FIGURA 4.29: CONFIGURACIÓN DE ESTÍMULOS EXTERNOS Y SIMULACIÓN DEL EL EJERCICIO 4.3.3.

4.3.4. Creación de melodías

Se pide implementar un sistema basado en un PIC16F877A trabajando a 4 MHz para integrarlo en una postal navideña. La postal cuenta con un interruptor conectado a RB0 que se abrirá o cerrará (1/0) al abrir y cerrar la postal respectivamente. Además, se cuenta con un zumbador conectado a RC2 para reproducir la melodía, que solo sonará cuando la postal esté abierta.

La melodía a reproducir se corresponde con la partitura inferior.
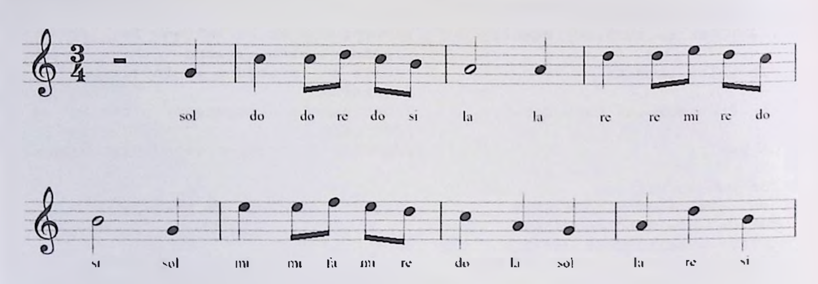
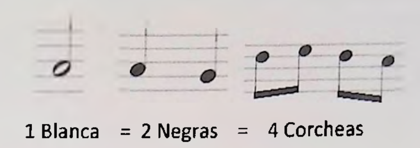


FIGURA 4.30: PARTITURA DE LA MELODÍA UTILIZADA EN EL EJERCICIO 4.3.4.

La frecuencia de cada una de las notas se encuentra en la tabla inferior.

Frecuencias (Hz)
Sol: 391,995
La: 440
Si: 493,883
Do: 523,25
Re: 587,33
Mi : 659,26
Fa : 689,46

La duración de cada nota se establecerá según la siguiente relación:



1 Blanca = 2 Negras = 4 Corcheas

Para una duración de corchea de 0.25 segundos. Se pide:

a) Implementa el código en C que satisfaga las condiciones del problema utilizando el módulo PWM del microcontrolador para la generación de las notas.

NOTA: La frecuencia de las notas se aproximarán al valor más cercano.

b) Indica el error de aproximación en frecuencia cometido para la nota la.

Solución:

a) Para la generación de las diferentes notas musicales se ajustarán diferentes señales PWM a la frecuencia de cada una. El ciclo de trabajo en todos los casos será del 50 % por lo que se obtendrá directamente desplazando el valor del periodo un bit hacia la derecha.

La duración de cada una de las notas se obtendrá utilizando una rutina de espera de 250 ms que se ejecutará el número de veces necesario para obtener la duración de la nota.

Configuración

El período T de la señal PWM se fija mediante el valor N depositado en el registro PR2. Así, se puede obtener el valor a asignar a PR2 para cada nota utilizando la fórmula:

$$T = (N + 1) * P * 1\mu s$$

donde P es el factor de división del pre-divisor del Timer2, P = 1, 4, 16.
Se escoge P = 16 ya que N > 255 y se obtienen los siguientes valores para cada nota:

Frec. (Hz)	PR2	CCPR1L	CCP1CON<5>
Sol: 391,995	158	79	0
La: 440	141	70	1
Si: 493,883	126	63	0
Do: 523,25	118	59	0
Re: 587,33	105	52	1
Mi : 659,26	94	47	0
Fa : 689,46	90	45	0

Para controlar la duración de cada una de las notas se utilizará el temporizador TIMER1 en modo temporizador para contabilizar 250 ms funcionando en modo interrupción.

Código fuente

```
#include <htc.h>

__CONFIG(FOSC_HS & WDTE_OFF & LVP_OFF & PWRTE_ON);
#define _XTAL_FREQ 4000000

// Definición de los valores de PR2 para cada nota
#define Sol 158
#define La  141
#define Si  126
#define Do  117
#define Re  105
#define Mi  94
#define Fa  90
#define Silencio 0
#define Fin 0xFF

//Definición de la duración de cada tipo de nota (múltiplo de 0.25 segundos)
#define Corchea 1
#define Negra   2
#define Blanca  4

void Pulsador(void);
void Tempor(void);
void Ton_pwm(void);
void Inic_pwm(void);

unsigned char TablaP, Tempo;
unsigned int Tiempo1=60000;        //Ajuste del Tiempo
//Tabla con la partitura
const char Notas[60]={Silencio,Blanca,Sol,Negra,Do,Negra,Do,Corchea,
                  Re,Corchea,Do,Corchea,Si,Corchea,La,Blanca,La,Negra,
                  Re,Negra,Re,Corchea,Mi,Corchea,Re,Corchea,Do,
                  Corchea,Si,Blanca,Sol,Negra,Mi,Negra,
                  Mi,Corchea,Fa,Corchea,Mi,Corchea,Re,Corchea,
                  Do,Negra,La,Negra,Sol,Negra,La,Negra,
                  Re,Negra,Si,Negra,Do,Blanca, Fin};

int main(){
    INTCON=0x90;        //Habilita interrupciones globales e interrupción externa
    while(1){
    while(PORTBbits.RB0==1);        //Comprueba el estado de la tarjeta ON/OFF
    INTCONbits.PEIE=0;              //Deshabilita interrupciones de periféricos
    T1CON=0;                        //para el TIMER1, el TIMER2 y el Módulo CCP1
    T2CON=0;
    CCP1CON=0;
        }
}
void interrupt ISR(void){
    if(INTCONbits.INTF==1) Pulsador();          //Comprueba si ha saltado la
                                                //interrupción externa
    else if(PIR1bits.TMR1IF==1) Tempor();       //Comprueba si ha saltado la
                                                //interrupción TIMER1
}
void Pulsador(void){
    INTCONbits.INTF=0;                  //Baja el flag de interrupción
    if(PORTBbits.RB0==0)return;         //Comprueba el pulsador
    Inic_pwm();                         //programar módulo CCP1 con la primera
                                        //nota
}
```

```
void Tempor(void){
    PIR1bits.TMR1IF=0;          //Baja el flag de interrupción del TIMER1
    TMR1=Tiempo1;               //Recarga el valor de los contadores del TIMER1
                                //para volver a contar 0.25s
    if(--Tempo==0) Ton_pwm();       //Decrementa el contador del tiempo de nota
}
//****************************************************************************
//Inic_pwm: Programar el módulo CCP1 en modo PWM, e iniciar la melodía.
//****************************************************************************
void Inic_pwm(void){
    T2CON=0x02;         //Programar post-divisor = 1, Timer2 detenido, y pre-
                        //divisor = 16.
    T1CON=0x30;         //Programar predivisor = 8, Timer1 detenido
    TMR1=Tiempo1;       //Carga el valor de los contadores del TIMER1 para contar
                        //0.25s
//    TMR1=100;         //Carga el valor de los contadores del TIMER1 para contar
                        //0.25s

   CCP1CON=0;           //Reset al módulo CCP1
    CCPR1L=0;           //Poner a 0 el ciclo de trabajo (0%)
    TablaP=0;           //Reinicia el apuntador de la tabla de notas a la primera
                        //posición
    TRISCbits.TRISC2=0;       //Poner el terminal CCP1 como salida.
    INTCONbits.PEIE=1;        //Habilita interrupciones de periféricos
    PIE1bits.TMR1IE=1;        //Habilita interrupción del TIMER1.
    CCP1CON|=0x0C;            //El módulo CCPI en modo PWM
      Ton_pwm();
    T2CONbits.TMR2ON=1;       //Arranca el TIMER2
    T1CONbits.TMR1ON=1;       //Arranca el TIMER1
}
//****************************************************************************
//TON_pwm: Ajusta el periodo de la señal PWM según el valor depositado en la
//TABLA NOTAS
//con un ciclo de trabajo del 50% y obtiene la duración de la nota en función
//del número
//de desbordamientos del TIMER1 en la variable TEMPO
//****************************************************************************
void Ton_pwm(void){
    CCP1CON=0;                              //Reset al módulo CCP1
    CCPR1L=0;                               //Poner a 0 el ciclo de trabajo (0%)
    if(Notas[TablaP]==0xFF) TablaP=0;       //Obtiene el valor de PR2
    CCPR1L=Notas[TablaP]/2;                 //Guardar PR2/2 en CCPR1L
      CCP1CONbits.CCP1X=Notas[TablaP]%2;    //Guarda el resto en el bit CCP1X
    PR2=Notas[TablaP];                      //Ajusta el periodo de la primera nota
    CCP1CON|=0x0C;                          //Ajusta el módulo CCP1 en modo PWM con
                                            //CCP1X:CCP1Y en 0 (M2=0).
    Tempo=Notas[++TablaP];                  //Obtiene la duración de la nota
      TablaP++;
}
```

c) El error de aproximación en frecuencia cometido en la nota la vendrá dado por el error de aproximación introducido al calcular el valor de PR2. Así pues, para el valor de PR2 calculado para la nota la se obtiene:

$$T_{LA\text{-}PWM} = (141+1) * 16 * 1\mu s = 2272\ \mu s \Rightarrow f_{LA\text{-}PWM} = 440.14\ Hz$$

Lo que equivale a un error de 0.032 %.



5. Módulos analógicos

5.1. Módulo generador de tensión de referencia (CVref)

El módulo generador de tensión de referencia integrado en el microcontrolador PIC16F877A consiste en un divisor de tensión resistivo en escalera de dieciséis escalones conectado a un multiplexor analógico. Este módulo proporciona un valor de tensión de referencia ajustable que puede ser utilizado como referencia V_{REF-} para el comparador analógico digital, como entrada V_{IN+} por los comparadores analógicos o como tensión de salida de referencia por RA2/AN2 siempre y cuando se utilice como salida conectada a una carga de alta impedancia, aunque está realmente pensado para utilizarse simplemente como comprobación del valor de tensión entregado al comparador. El módulo CVref proporciona dos rangos de ajuste de tensión y puede ser apagado en caso de no ser utilizado para reducir el consumo de energía. La configuración de este módulo se realiza mediante el registro CVRCON (A2.12).

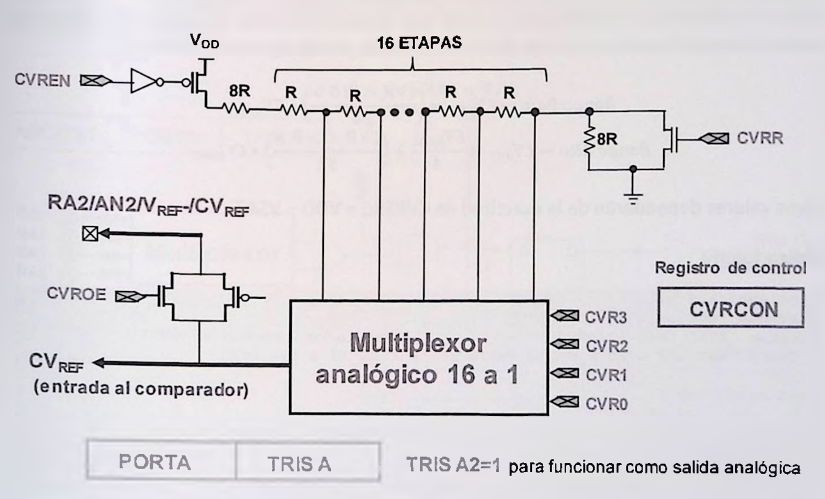


FIGURA 5.1: ESQUEMA DE BLOQUES DEL MÓDULO CVREF.

5.1.1. Generación de señal analógica

Utiliza el módulo generador de tensión de referencia para generar una señal diente de sierra.

Solución:

Para generar la señal diente de sierra se hará uso del generador de tensión de referencia integrado en el microcontrolador PIC16F877A.

Configuración

La configuración del módulo de tensión de referencia se realizará ajustando el registro CVRCON (A2.12). Para facilitar el ajuste del módulo se definirán varias constantes que harán referencia al modo de funcionamiento del módulo y que se utilizarán mediante la función 'OR' como argumento de una macro. La definición de las constantes y de la macro se incluirán en el archivo de definiciones "analog.h" y quedarán de la siguiente manera:

```
#define CVR_ON 0x80                                 //Módulo habilitado
#define CVR_OFF 0x00                                //Módulo deshabilitado
#define CVR_LOW 0x20                                //Rango bajo de funcionamiento
#define CVR_HIGH 0x00                               //Rango alto de funcionamiento
#define CVR_OUT_ON 0x40                             //Salida CVREF habilitada por RA2
#define SET_CVREF(MODE) CVRCON=MODE                 //Función de configuración
```

Para la generación de la tensión de referencia habrá que tener en cuenta también el tiempo de respuesta definido por el fabricante, en este caso 10 µs. Los valores de tensión de referencia vendrán dados en función del modo de funcionamiento por las siguientes fórmulas:

$$Rango\ Bajo \rightarrow CV_{REF} = \left(\frac{CVR < 3{:}0 >}{24}\right) * CV_{RSRC}$$

$$Rango\ Alto \rightarrow CV_{REF} = \frac{CV_{RSRC}}{4} + \left(\frac{CVR < 3{:}0 >}{32}\right) * CV_{RSRC}$$

Cuyos valores dependerán de la exactitud de CVRSRC = VDD – VSAT.

Código fuente

```
#include <htc.h>
#include "analog.h"
#define _XTAL_FREQ 4000000              //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

void main(void){
        unsigned char i;
        di();                           //Se deshabilitan interrupciones
        while(1){
           for (i=0;i<16;i++){
                SET_CVREF(CVR_ON | CVR_LOW | CVR_OUT_ON | i);  //Ajuste CVREF
                __delay_us(10);
           }
        }
}
```

Modifica el programa de forma que permita conmutar entre un ajuste en rango alto y un ajuste en rango bajo. Este modo de funcionamiento se conmutará mediante un pulsador conectado a RB0.

Modifica el programa de forma que se muestre la tensión generada con dos decimales y el modo "fino/grueso" de ajuste de la tensión de salida en una pantalla LCD.

5.2. Módulos Comparadores de Tensión Analógica

El microcontrolador PIC16F877A contiene dos comparadores analógicos cuyas entradas están multiplexadas con los terminales RA0-RA3 y las salidas con los terminales RA4 y RA5. Además, el módulo generador de tensión de referencia también puede utilizarse como referencia de tensión en la entrada no inversora de los comparadores. El modo de funcionamiento de los comparadores viene determinado por el valor almacenado en el registro CMCON (A2.13).

El flag de interrupción, bit CMIF del registro PIR2, asociado a los módulos comparadores se activará siempre que haya un cambio en el valor de salida de cualquiera de los comparadores por lo que será necesario registrar la información de salida de los comparadores para poder determinar el cambio que se ha originado.

En el caso de que se desee que se genere una interrupción cuando cambie el valor a la salida de los comparadores se deberá activar el bit CMIE del registro PIE2 y los bits PEIE y GIE del registro INTCON.

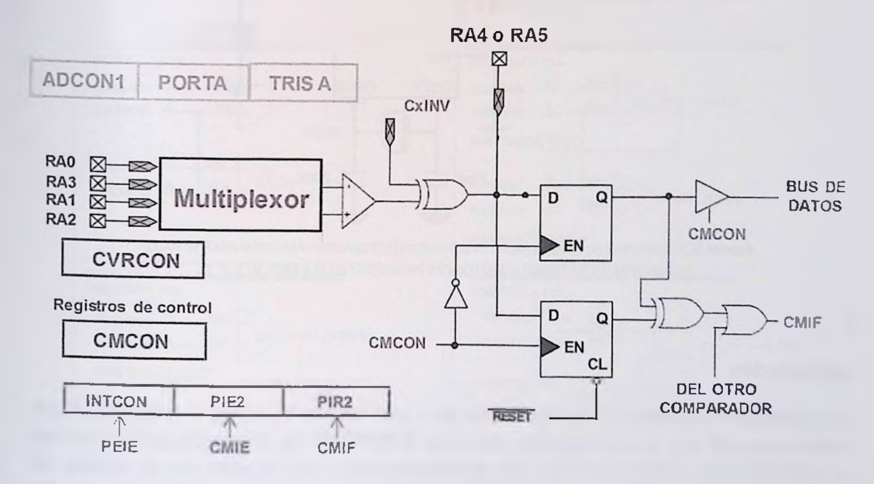


FIGURA 5.2: ESQUEMA DE BLOQUES DEL MÓDULO COMPARADOR ANALÓGICO.

5.2.1. Comparar dos señales analógicas

Realiza un programa que compare dos señales analógicas A y B (0-5V) y active un led en caso de A>B.

a) Indica las conexiones necesarias con el microcontrolador.

b) Implementa el programa que cumpla las especificaciones indicadas.

Solución:

a) La solución propuesta va a utilizar uno de los comparadores analógicos integrados en el microcontrolador PIC16F877A. Para ello se conectarán las señales analógicas A y B a las entradas de uno de los dos comparadores. Para el funcionamiento del microcontrolador también será necesario conectar correctamente las entradas VDD y VSS a +5V y 0V respectivamente además de un oscilador, en este caso un oscilador de cristal a 4 MHz, entre las entradas OSC1 y OSC2. Por último, se conectará un LED con su resistencia para limitar la corriente al terminal RB0 tal y como aparece en la figura inferior.

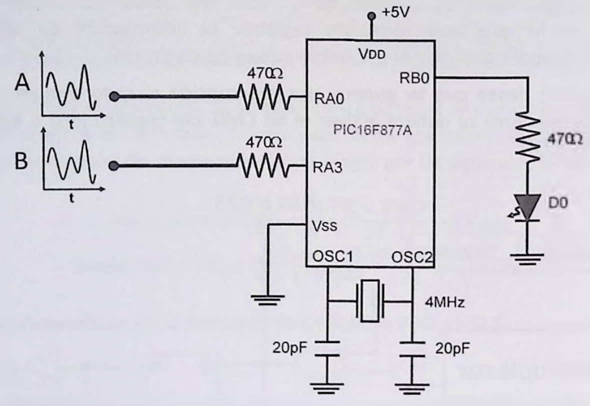


FIGURA 5.3: ESQUEMA DE CONEXIONES CON EL MICROCONTROLADOR PARA COMPARAR EL VALOR DE DOS SEÑALES DE ENTRADA ANALÓGICAS PROPUESTO EN EL EJERCICIO 5.2.1.

b)

Configuración

Para configurar el módulo comparador analógico será necesario ajustar el registro CMCON (véase anexo A2.13). Esto se realizará mediante la definición de diferentes constantes que se utilizarán para definir el modo de funcionamiento y se incluirán en el archivo de definiciones "analog.h". Las constantes harán referencia a la configuración de los comparadores del recuadro inferior y serán las siguientes:

```
#define CA_RESET                    0x00
#define CA_A0_A3_NC_NC_ON_A4        0x01
#define CA_A0_A3_A1_A2              0x02
#define CA_A0_A3_A1_A2_ON_A4_A5     0x03
#define CA_A0_A3_A1_A3              0x04
#define CA_A0_A3_A1_A3_ON_A4_A5     0x05
#define CA_A0_VR_A1_VR              0x06
#define CA_A3_VR_A2_VR              0x0E
#define CA_NC_NC_NC_NC_OFF          0x07
#define CA_C1INV                    0x10
#define CA_C2INV                    0x20
```

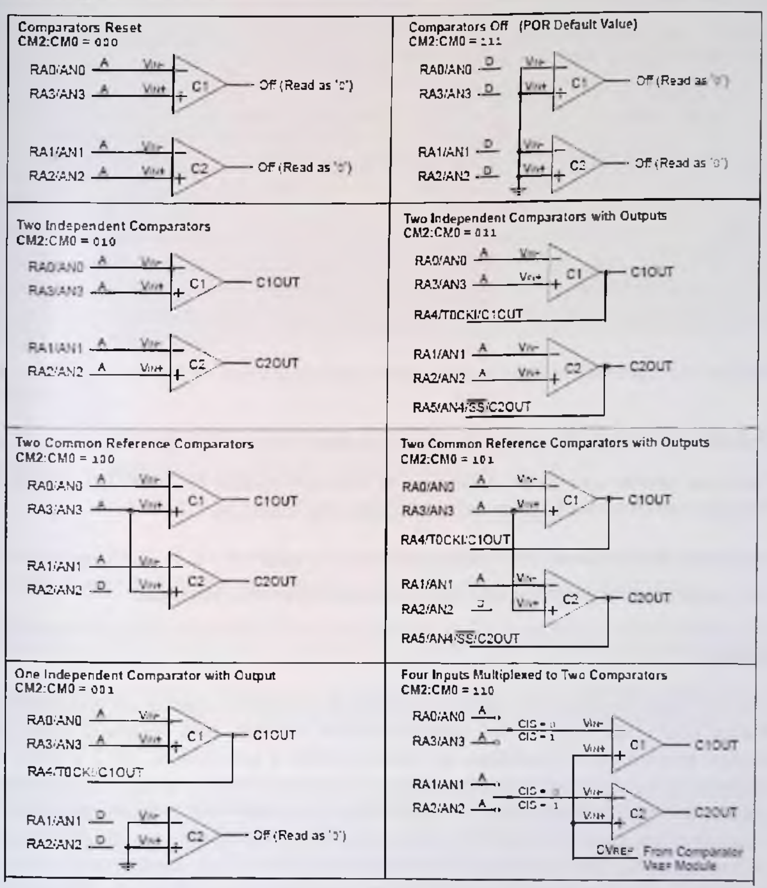


FIGURA 5.4: RESUMEN DE LOS MODOS DE FUNCIONAMIENTO DE LOS COMPARADORES ANALÓGICOS.

Una vez definidas las constantes se utilizará una macro, que también se incluirá en el archivo de definiciones "analog.h", para ajustar la configuración del módulo. La macro tendrá la siguiente estructura:

```
#define SETUP_COMPARATOR(CONFIG_COMPARATOR) \
      CMCON=CONFIG_COMPARATOR;
```

Código fuente

```
#include <htc.h>
#include "analog.h"

#define _XTAL_FREQ 4000000                    //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

void main(void){
      TRISB=0xFE;                        //Terminal RB0 como salida
      di();                              //Se deshabilitan interrupciones
      SETUP_COMPARATOR(CA_A0_A3_NC_NC_ON_A4|CA_C1INV);    //Configura el
                                                          //comparador 1 con
                                                          //entradas por RA0 y
                                                          //RA3 y salida
                                                          //invertida por RA4
       while(1){
          if(C1OUT) RB0=1;
          else      RB0=0;
       }
}
```

Modificar el programa para que el led se active cuando la señal en B es mayor que en A.

5.2.2. Comparación de una señal analógica con una señal de referencia

Se propone diseñar un sistema basado en un microcontrolador PIC16F877A trabajando a 4 MHz que encienda un LED si una señal analógica, V_{IN1} supera los 2.2 V.

a) Indica las conexiones necesarias con el microcontrolador.

b) Implementa el programa que cumpla las especificaciones indicadas.

Solución:

a) Para el funcionamiento del microcontrolador es necesario conectar correctamente las entradas VDD y VSS a +5 V y 0 V respectivamente. Además, será necesario conectar el oscilador, en este caso un oscilador de cristal a 4 MHz a las entradas OSC1 y OSC2 tal y como aparece en la figura 5.5. Por otro lado, se conectará la señal analógica V_{IN} a una de las entradas del módulo comparador RA0/AN0/C1IN-, la salida del módulo generador de tensión de referencia RA2/AN2/CVREF a la otra entrada del comparador 1 RA3/AN3/C1IN+ y el LED a la salida correspondiente del comparador RA4/C1OUT. Dado que la salida del comparador RA4/C1OUT es en drenador abierto será necesario utilizar una resistencia de PULL-UP conectada a +5 V y al ánodo del LED según se muestra en la siguiente figura.

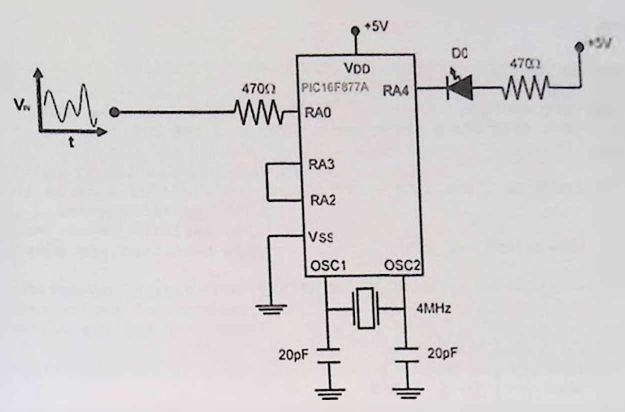


FIGURA 5.5: ESQUEMA DE CONEXIONES CON EL MICROCONTROLADOR PARA COMPARAR EL VALOR DE UNA SEÑAL DE ENTRADA ANALÓGICA CON UN VALOR DE REFERENCIA PROPUESTO EN EL EJERCICIO 5.2.2.

b) Para su correcto funcionamiento el programa deberá configurar el módulo generador de tensión de referencia para obtener una tensión de 2.2 V. Por otro lado, se configurará el módulo comparador C1 con entradas analógicas por RA0/AN0/C1IN- y RA3/AN3/C1IN+ y su salida RA4/C1OUT habilitada y conectada a la salida del módulo generador de tensión de referencia RA2/AN2/CVREF.

Configuración

Para la generación de 2.2 V se utilizará el módulo generador de tensión de referencia trabajando en rango alto y se aplicará la siguiente fórmula:

$$CV_{REF} = \frac{CV_{RSRC}}{4} + \left(\frac{CVR < 3{:}0 >}{32}\right) * CV_{RSRC}$$

Si se sustituye $CV_{REF} = 2.2$ V y $CV_{RSRC} = 4.4$ V y se despeja se obtiene que CVR = 8. Una vez obtenido el valor de CVR se configura el módulo utilizando la siguiente instrucción:

```
SET_CVREF(CVR_ON | CVR_HIGH | CVR_OUT_ON | 8);        //Ajuste CVREF=2.2V con
                                                      //salida habilitada
```

Por otro lado, se deberá configurar el módulo comparador 1 con entradas analógicas por RA0/AN0/C1IN- y RA3-/AN3/C1IN+ y salida RA4/C1OUT habilitada. Se utilizarán para ello las siguientes instrucciones.

```
TRISA4=0;                                //Se habilita la salida por RA4
set_adc_inputs(AN0_AN1_AN3);             //Se habilitan entradas analógicas AN0,
                                         //AN1 y AN3
SETUP_COMPARATOR(CA_A0_A3_NC_NC_ON_A4)   //Configura el comparador analógico 1
                                         //con entradas por RA0(-) y RA3(+)
                                         //y salida por RA4
```

Código fuente

```
#include <htc.h>
#include "analog.h"
#define _XTAL_FREQ 4000000                //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);
void main(void){
      di();                               //Se deshabilitan interrupciones
      SET_CVREF(CVR_ON | CVR_HIGH | CVR_OUT_ON | 8); //Ajuste de la tensión de
                                                     //referencia
      TRISA4=0;                                 //Se habilita la salida por RA4
      set_adc_inputs(AN0_AN1_AN3);              //Se habilitan entradas analógicas
AN0, AN1 y AN3
      SETUP_COMPARATOR(CA_A0_A3_NC_NC_ON_A4);   //Configura el comparador
                                                //analógico 1 con entradas por
                                                //RA0(-) y RA3(+)y salida por RA4
      while(1);
}
```

Modificar el programa para que se ilumine el led cuando la señal de entrada sea inferior a 0.5V.

5.2.3. Sensor de luminosidad TSL251RD

Se pretende realizar un programa basado en un microcontrolador PIC16F877A trabajando a 4MHZ que encienda un led cuando detecte que la luz ambiente se ha reducido por debajo de un nivel de referencia ajustable. Para ello se cuenta con el sensor de luminosidad TSL251RD que proporciona un valor de tensión de salida proporcional a la irradiancia recibida.

a) Indica las conexiones necesarias con el microcontrolador.

b) Implementa el programa que cumpla las especificaciones indicadas.

Solución:

a) Para el funcionamiento del microcontrolador es necesario conectar correctamente las entradas VDD y VSS a +5 V y 0 V respectivamente. Además, será necesario conectar el oscilador, en este caso un oscilador de cristal a 4MHz a las entradas OSC1 y OSC2 tal y como aparece en la figura 5.6.

El integrado TSL251RD se conectará siguiendo las indicaciones de la hoja de especificaciones con su salida analógica conectada a la entrada analógica AN1/RA1/C2IN- y la tensión de referencia analógica se conectará a la entrada analógica AN3/RA3. La salida del módulo comparador 2 RA5/AN4/C2OUT se conectará a un LED utilizando una resistencia que limitará la corriente del diodo cuyo cátodo estará conectado a 0 V como se muestra en la figura siguiente.

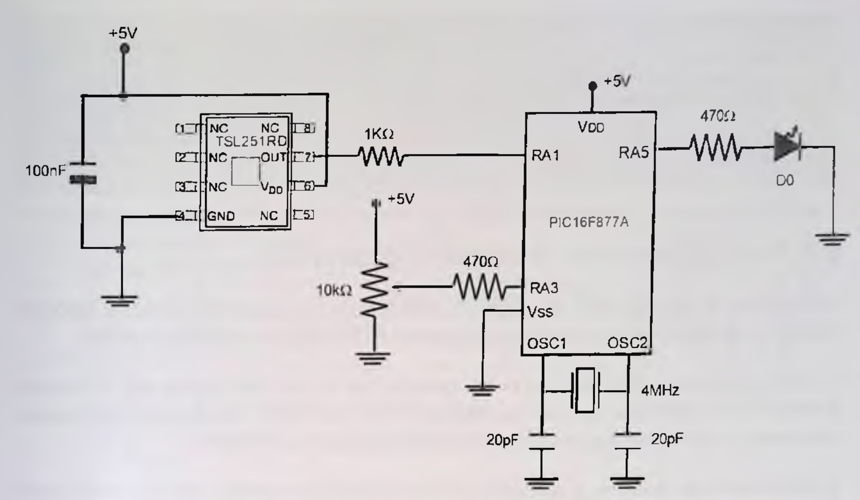


FIGURA 5.6: ESQUEMA DE CONEXIONES CON EL MICROCONTROLADOR PARA UTILIZAR EL SENSOR DE LUMINOSIDAD TSL251RD PROPUESTO EN EL EJERCICIO 5.2.3.

b) Para la realización del sistema será necesaria la utilización de uno de los comparadores analógicos de los que dispone el PIC16F877A con sus entradas conectadas a las señales analógicas (salida del TSL251RD y tensión de referencia del potenciómetro). En este caso se utilizará el comparador 2 como se observa en las conexiones de la figura superior con su salida habilitada por RA5, que permitirá de forma automática encender el LED cuando la tensión de salida del sensor de luminosidad supere el valor de referencia.

Configuración

La configuración del módulo comparador se realizará habilitando el comparador 2 y con sus entradas analógicas por RA1/AN1/C2IN- y RA3/AN3 y su salida digital asociada RA5/AN4/C2OUT mediante las siguientes instrucciones:

```
TRISA5=0;                                //Se habilita la salida por RA5
set_adc_inputs(AN0_AN1_AN3);             //Se habilitan entradas analógicas
                                         //AN0, AN1 y AN3
SETUP_COMPARATOR(CA_A0_A3_A1_A3_ON_A4_A5) //Configura el comparador
                                         //analógico 2 con entradas por RA1(-)
                                         //y RA3(+) y salida por RA5
```

Código fuente

```
#include <htc.h>
#include "analog.h"

#define _XTAL_FREQ 4000000                     //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);
```

```
void main(void){
      di();                              //Se deshabilitan interrupciones
      TRISA5=0;                          //Se habilita la salida por RA5
      set_adc_inputs(AN0_AN1_AN3);       //Se habilitan entradas analógicas
      SETUP_COMPARATOR(CA_A0_A3_A1_A3_ON_A4_A5);     //Configura el comparador 2
                                                     //con entradas por RA1(-) y
                                                     //RA3(+) y salida por RA5
      while(1);
}
```

5.3. Módulo Conversor Analógico Digital (CAD)

Los registros de configuración asociados al módulo CAD son ADCON0 (A2.10) y ADCON1 (A2.11) y el resultado de la conversión se almacena en los registros ADRESH y ADRESL.

El CAD integrado en el PIC16F877A es un conversor de 10 bits que cuenta con 8 entradas analógicas (AN0-AN7) que se reparten entre el PORTA y el PORTE. La selección del canal de conversión se realiza mediante los bits CHS2:CHS0 del registro ADCON0.

El amplificador de muestreo y retención está compuesto básicamente por un condensador (sin amplificadores de entrada ni de salida), que empieza a cargarse en cuanto se selecciona en el multiplexor el canal deseado. La tensión en el condensador sigue la evolución de la tensión de entrada, y cuando se da una orden el condensador se desconecta de la entrada analógica y empieza la conversión. La tensión de referencia para la conversión A/D puede ser la tensión de alimentación del microcontrolador (valor por defecto) o una tensión externa que se aplique entre los terminales de referencia AN3/VREF+ y AN2/VREF-. Esta selección se realiza mediante los bits PCFG3:PCFG0 del registro ADCON1. Las conversiones A/D se realizan en sincronismo con una señal de reloj. Este reloj se obtiene o bien del oscilador principal del microcontrolador mediante un divisor programable, o bien de un oscilador RC interno de frecuencia fija. Para que el CAD funcione mientras el microcontrolador está en el modo de bajo consumo (*sleep*), hay que seleccionar el oscilador RC interno.

Para iniciar una conversión A/D hay que activar el bit de control GO/DONE#. Cuando el resultado de la conversión está ya disponible en ADRESH y ADRESL se desactiva automáticamente el bit de estado GO/DONE# y se activa el bit ADIF del registro PIR1 para solicitar interrupción. Si el bit ADIE del registro PIE1 está activo y el sistema de interrupción del microcontrolador está habilitado el microcontrolador recibe una solicitud de interrupción.

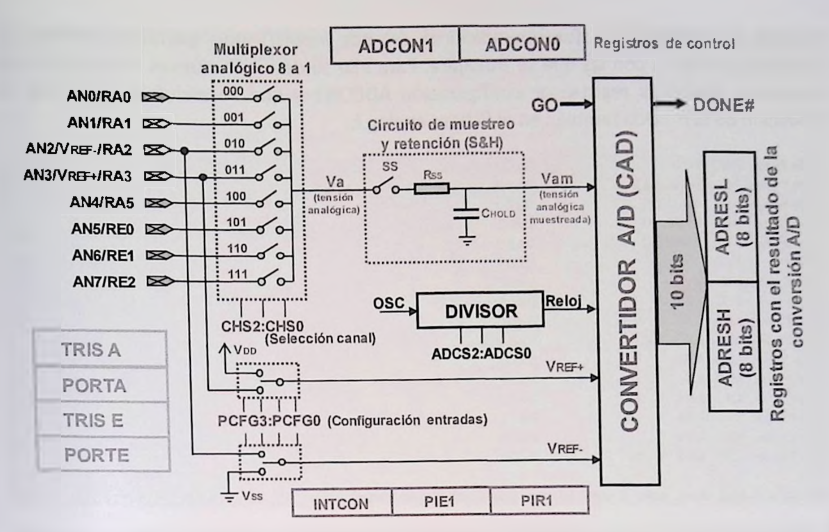


FIGURA 5.7: ESQUEMA DE BLOQUES DEL MÓDULO CONVERSOR ANALÓGICO DIGITAL.

5.3.1. Configuración del módulo CAD

En este ejemplo se pretende crear una función que configure el módulo CAD de un PIC16F877A para trabajar con un oscilador a 4 MHz, utilizando todas las entradas analógicas disponibles.

Solución:

La solución propuesta va a considerar el caso del microcontrolador trabajando a 4 MHz utilizando todas las entradas analógicas pero se extenderá para un cualquier caso genérico de configuración del módulo CAD pudiendo utilizarse para cualquier configuración del módulo a partir de los parámetros que se le pasen a las funciones.

Configuración

Para la creación de la función será necesario definir diferentes constantes que determinen el modo de trabajo del módulo y que permitan ajustar los registros de configuración ADCON0 y ADCON1. Así, se definirá la constante ADCLK_FREQ_DIV que tomará el valor de la constante _XTAL_FREQ dividida por 1.25 MHz de forma que permita obtener el factor divisor a aplicar al reloj principal en el caso de que el CAD trabaja con dicho reloj y cumplir con las especificaciones de funcionamiento del fabricante. También se definirá la constante ADCLK_RC que seleccionará el reloj interno del CAD. Estas constantes junto con la definición de la función *setup_adc* se incluirán en el fichero analog.h que tendrá la siguiente estructura.

```
#define ADC_CLK_RC 0xC1
#define ADC_FREQ_DIV (float)(_XTAL_FREQ/1250000)
extern void setup_adc(unsigned char mode);
```

Además se utilizará una función adicional *set_adc_inputs()* que permitirá definir las entradas analógicas con las que se trabajará. Para ello se definirán nuevas constantes que permitirán ajustar el registro de configuración ADCON1 y que se incluirán junto con la definición de la función también en el fichero analog.h.

```
#define ADC_OFF                  0x00
#define ALL_DIGITAL              0x06
#define ALL_ANALOG               0x00
#define ALL_ANALOG_V             0x01
#define ALL_ANALOG_V_V-          0x08
#define AN0                      0x0E
#define AN0_V_V-                 0x0F
#define AN0_AN1_V                0x05
#define AN0_AN1_V_V-             0x0D
#define AN0_AN1_AN3              0x04
#define AN0_AN4                  0x02
#define AN0_AN4_V                0x03
#define AN0_AN4_V_V-             0x0C
#define AN0_AN5                  0x09
#define AN0_AN5_V                0x0A
#define AN0_AN5_V_V-             0x0B

extern void set_adc_inputs(unsigned char analog);
```

El código de las funciones *setup_adc()* y *set_adc_inputs()* se incluirá en el fichero analog.c que deberá añadirse dentro del proyecto en el que se utilice el módulo CAD así como la sentencia #include "analog.h". como se muestra en el ejemplo siguiente.

Código fuente

```
//Ajusta las entradas analógicas/digitales
extern void set_adc_inputs(unsigned char analog){   //Selecciona las entradas
                                                    //analógicas
        ADCON1=(ADCON1&0xF0)|analog;
}

//Selecciona el modo de reloj del CAD, ajusta el canal 0, enciende el módulo y
//ajusta a la izquierda
extern void setup_adc(unsigned char mode){     //Selecciona el modo de reloj
      if (mode==0x00){                         //CAD apagado
        ADCON0=0x00;
        ADCON1&=0x06;                          //E/S digitales
      }
      else if (mode==0xC1){                    //Reloj RC
        ADCON0=0xC1;                           //con ajuste derecho
        ADCON1&=0xBF;
      }
      else{
        if(ADC_FREQ_DIV<=1){        //Fosc <= 1.25MHz
                ADCON0=0x01;        //ADCS1=0, ADCS0=0, Canal 0,
                                    //Módulo encendido
                ADCON1&=0xBF;       //ADFM=1(ajuste a la derecha), ADCS2=0
        }
        else if(ADC_FREQ_DIV<=2){        //Fosc <= 2.5MHz
                ADCON0=0x01;             //ADCS1=0, ADCS0=0, Canal 0,
                                         //Módulo encendido
                ADCON1|=0xC0;            //ADFM=1(ajuste a la derecha), ADCS2=1
        }
```

```
        else if(ADC_FREQ_DIV<=4){         //Fosc <= 5MHz
                ADCON0=0x41;              //ADCS1=0, ADCS0=1, Canal 0,
                                          //Módulo encendido
                ADCON1&=0xBF;             //ADFM=1(ajuste a la derecha), ADCS2=0
        }
        else if(ADC_FREQ_DIV<=8){         //Fosc <= 10MHz
                ADCON0=0x41; ;            //ADCS1=0, ADCS0=1, Canal 0,
                                          //Módulo encendido
                ADCON1|=0xC0;             //ADFM=1(ajuste a la derecha), ADCS2=1
        }
        else if(ADC_FREQ_DIV<=16){        //Fosc <= 20MHz
                ADCON0=0x81;              //ADCS1=1, ADCS0=0, Canal 0,
                                          //Módulo encendido
                ADCON1&=0xBF;             //ADFM=1(ajuste a la derecha), ADCS2=0
        }
        else{                             //Fosc <= 40MHz
                ADCON0=0x81;              //ADCS1=1, ADCS0=0, Canal 0,
                                          //Módulo encendido
                ADCON1|=0xC0;             //ADFM=1(ajuste a la derecha), ADCS2=1
        }
    }
}
```

5.3.2. Conversión analógica digital en un canal

Se pretende utilizar el módulo CAD para realizar una conversión del valor de tensión (0-5 V) a la entrada del puerto RA0 y que muestre el valor de los cuatro bits más significativos obtenidos de la conversión en los led conectados a las terminales del puerto B (RB3-RB0).

a) Indica las conexiones necesarias con el microcontrolador.

b) Implementa el programa que cumpla las especificaciones indicadas.

Solución:

a) En la figura inferior se muestran las conexiones necesarias con el microcontrolador.

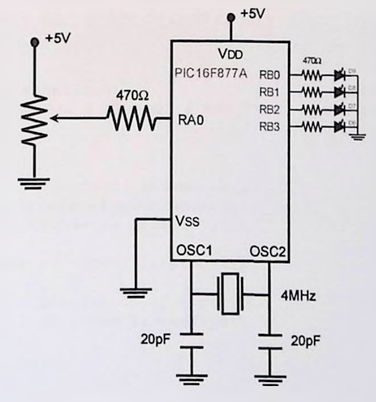


FIGURA 5.8: ESQUEMA DE CONEXIONES CON EL MICROCONTROLADOR PARA CONVERTIR UNA SEÑAL DE ENTRADA ANALÓGICA.

b)

Configuración

Una vez configurado el módulo CAD en el canal 0 utilizando la función del ejemplo anterior *setup_adc()* y habiendo esperado el tiempo de adquisición es necesario lanzar la conversión poniendo a '1' el bit GO del registro ADCON0. Para esto se definirá la macro *START_ADC_CONV*. Por último se creará una función que recoja el valor obtenido tras la conversión *read_adc()*.

```
#define START_ADC_CONV GO=1        //Lanza una conversión analógico digital
extern unsigned int read_adc(void);
```

La estructura de la función anterior y la macro se incluirá dentro del fichero analog.h y el código correspondiente que aparece a continuación en el fichero analog.c.

```
// Recoge el valor de la conversión de 10 bits
extern unsigned int read_adc(){
      while(GO)continue;                //espera a que finalice la conversión
      return ((ADRESH<<2)+(ADRESL>>6));
}
```

Una vez creada la función anterior junto con las que se mostraron en el ejemplo anterior solo será necesario ir llamándolas una tras otra desde el programa principal para obtener el valor de la conversión. Es necesario también un último detalle que consiste en configurar el puerto B como salidas ya que por defecto se configurarán como entradas al arrancar el microcontrolador. Para ello basta con utilizar la instrucción:

TRISB = 0x00;

Código fuente

```
#include <htc.h>
#include "analog.h"

#define _XTAL_FREQ 4000000                    //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

void main(void){
      int x;
      di();                          //Se deshabilitan interrupciones
      TRISB=0x00;                    //Puerto B configurado como salida
      setup_adc(ADC_CLK_OSC);        //Inicializa el módulo CAD en el canal 0
      set_adc_inputs(AN0);
      __delay_us(50);                //Espera el tiempo de adquisición (aprox 44us)
      while(1){
        START_ADC_CONV;              //Lanza la conversión
        x=read_adc();                //Obtiene el valor de la conversión
        PORTB = (x>>6);
      }
}
```

5.3.3. Conversión analógica digital de varios canales

En este ejemplo se pretende utilizar el módulo CAD para realizar la conversión de 8 señales analógicas y almacenar los valores obtenidos en una tabla que se actualice de forma continua con los nuevos valores.

Solución:

En el ejemplo anterior no era necesario seleccionar el canal ya que se trabajaba con el canal 0, que es el que se establece por defecto al arrancar el módulo, pero ahora será necesaria una función que permita seleccionar el canal a convertir de entre las 8 entradas posibles.

Configuración

Para ello se definirá en "analog.h" la estructura de una nueva función que permita seleccionar el canal analógico *set_adc_channel():*

```
extern void set_adc_channel(unsigned char channel);
```

El código de la función anterior se incluirá en el archivo "analog.c" y lo que hará será modificar el registro ADCON0 para seleccionar el canal según se muestra a continuación:

```
//Selecciona el canal de conversión
//Cada vez que se cambia de canal es necesario esperar
//el tiempo de adquisición (aprox 44us) para que
//se estabilice la tensión a la entrada del CAD
//antes de lanzar una conversión nueva.

extern void set_adc_channel(unsigned char channel){
      ADCON0 = (ADCON0&0xC7)| ((channel%8)<<3);
}
```

De esta manera únicamente será necesario crear un array de tipo int para ir almacenando los valores obtenidos tras la conversión de cada canal y utilizar un contador que indique el canal a convertir en cada momento.

Código fuente

```
#include <htc.h>
#include "analog.h"

#define _XTAL_FREQ 4000000              //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

void main(void){

      unsigned char i;
      int tabla[8];
      i=0;
      di();                             //Se deshabilitan interrupciones
      setup_adc(ADC_CLK_OSC);           //Inicializa el módulo CAD
      set_adc_inputs(ALL_ANALOG);
```

```
        while(1){
        set_adc_channel(i%8);
        __delay_us(50);              //Espera el tiempo de adquisición
        START_ADC_CONV;              //Lanza la conversión
        tabla[i%8]=read_adc();       //Obtiene el valor de la conversión
        i++;
        }
}
```

5.3.4. Sensor de Temperatura TC1047A

Se pretende realizar un programa basado en el microcontrolador PIC16F877A trabajando a 4 MHz que recoja el valor de temperatura proporcionado por el sensor TC1047A (consultar nota de aplicación de Microchip© DS00938A para más información sobre el funcionamiento del sensor) y lo muestre en la pantalla LCD HITACHI HD44780.

a) Indica las conexiones necesarias con el microcontrolador.

b) Implementa el programa que cumpla las especificaciones indicadas.

Solución:

a) Para el funcionamiento del microcontrolador será necesario conectar correctamente las entradas VDD y VSS a +5 V y 0 V respectivamente y el oscilador, en este caso un oscilador de cristal a 4 MHz, a las entradas OSC1 y OSC2 tal y como aparece en la figura inferior. El sensor de temperatura TC1047A se conectará siguiendo las indicaciones de la hoja de especificaciones con su salida conectada a una de las entradas analógicas del microcontrolador (RA0/AN0). La conexión de la pantalla LCD se realizará siguiendo las instrucciones del fabricante para utilizar el modo de comunicaciones de 4 bits a través de los terminales RD0-RD4.

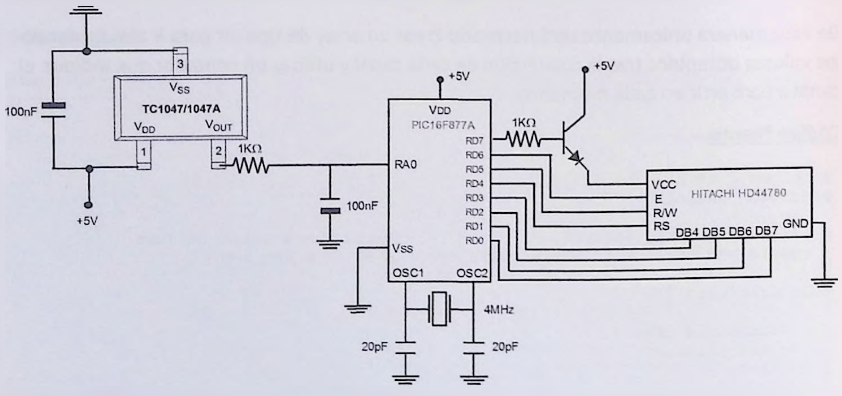


FIGURA 5.9: ESQUEMA DE CONEXIONES CON EL MICROCONTROLADOR PARA UTILIZAR EL SENSOR DE TEMPERATURA ANALÓGICO TC1047 PROPUESTO EN EL EJERCICIO 5.3.4.

b) Para realizar el programa será necesario configurar el módulo CAD con entrada analógica por RA0 además de definir los terminales del puerto D como salidas. El manejo de la pantalla LCD se realizará utilizando las funciones definidas en la librería "lcd.h".

Configuración

El módulo CAD se configurará con entrada analógica por RA0 y utilizando el reloj interno del microcontrolador mediante las siguientes instrucciones:

```
setup_adc(ADC_CLK_OSC);              //Inicializa el módulo CAD
set_adc_inputs(AN0);                 //Configura las entradas analógicas
set_adc_channel(0);                  //Selecciona el canal de entrada del conversor
```

Para representar el valor de temperatura en la pantalla LCD será necesario convertir el valor de tensión leído a la salida del sensor a °C utilizando la siguiente fórmula:

$$VOUT = (10\ mV/°C)\ (Temperature\ °C) + 500\ mV$$

Una vez obtenido el valor de temperatura en °C se deberá representar en la pantalla LCD. Para ello se hará uso de la función printf incluida en la librería "stdio.h". Esta función se encarga de dar formato a una cadena de caracteres en código ascii y va llamando a la función *putch* para enviar cada uno de los caracteres de la cadena. En este caso se definirá la función *putch* de forma que lo que haga sea enviar un carácter a la pantalla LCD de la siguiente manera:

```
void putch(char x){
      lcd_putch(x);
}
```

Código fuente

```
#include <htc.h>
#include <stdio.h>
#include "analog.h"
#include "lcd.h"

#define _XTAL_FREQ 4000000               //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

void putch(char);
void main(void){
      float temp;                        //Variable para almacenar la temperatura
      lcd_init();                        //Inicialización de la pantalla LCD
      di();                              //Se deshabilitan interrupciones
      setup_adc(ADC_CLK_OSC);            //Inicializa el módulo CAD
      set_adc_inputs(AN0);               //Se configura RA0 como entrada
                                         //analógica
      set_adc_channel(0);                //Se selecciona el canal de entrada
                                         //analógico en RA0
      while(1){
         START_ADC_CONV;                 //Lanza la conversión
         temp=(float)read_adc();         //Recoge el valor de la conversión
         temp= ((temp*4.88)-500)/10;     //Convierte el valor a grados
```

```
        lcd_goto(0);                    //Posiciona el cursor al inicio de la
                                        //pantalla LCD
        printf("Temp = %.2f C  ",temp); //Muestra el valor de temperature por
                                        //la pantalla LCD
        __delay_ms(500);                //Espera medio Segundo antes de lanzar
                                        //otra conversión
      }
}

void putch(char x){
      lcd_putch(x);
}
```

6. Módulos de comunicaciones

En este capítulo se presentan los módulos de comunicaciones serie que integra el microcontrolador PIC16F877A. En particular, el PIC16F877A integra dos módulos de comunicaciones serie independientes que utilizan los terminales asociados al PORTC. Estos módulos son el módulo USART (*Universal Synchronous Asynchronous Receiver Transmitter*) y el módulo MSSP (*Master Synchronous Serial Port*).

6.1. Módulo USART

El microcontrolador PIC16F877A integra un módulo transmisor/receptor asíncrono/síncrono universal (*Universal Synchronous Asynchronous Receiver Transmitter,* USART) también denominado como interfaz de comunicaciones serie (*Serial Communications Interface*, SCI). Este módulo se utiliza generalmente en modo *full-duplex* para comunicaciones asíncronas con ordenadores personales aunque también puede emplearse en modo *half-duplex* para comunicaciones síncronas con otros periféricos como memorias, conversores analógico/digitales, etc. (para más información sobre comunicaciones síncronas utilizando el módulo USART consultar las hojas de especificaciones del microcontrolador). Este módulo permite por tanto la comunicación del microcontrolador utilizando los protocolos RS232, RS422 y RS485 siempre y cuando se adapten los niveles de señal utilizados por el microcontrolador (generalmente 5V) a los utilizados por los protocolos anteriores mediante la utilización, por ejemplo, del integrado MAX232 o similar.

Dado que el módulo USART es un módulo periférico integrado en el microcontrolador, este requerirá para su funcionamiento de una configuración inicial al comienzo del programa. Una vez configurado, el módulo USART funcionará de forma autónoma requiriendo la atención del microcontrolador sólo cuando sea necesario ya sea mediante interrupciones, cuando estas estén habilitadas, o únicamente activando los indicadores "*flags*" asociados al módulo. La configuración de los módulos transmisor y receptor se realiza mediante los registros TXSTA (A2.14), RCSTA (A2.15) y SPBRG mientras que el envío y recepción de datos se realiza utilizando los registros TXREG y RCREG respectivamente.

Con respecto al módulo transmisor, el bit TXIF del registro PIR1 se pone a '0' cada vez que escribimos un nuevo dato sobre el registro TXREG y, pasa a '1' cuando el contenido de este buffer se copia sobre el TSR para dar inicio a la transmisión del nuevo carácter. En este momento, si se desea, se puede provocar una interrupción al controlador. El bit TRMT del registro TXSTA, indica el estado del registro de desplazamiento TSR. Se pone a '1' cuando se encuentra vacío, sin nada que transmitir.

Pueden realizarse transmisiones de 9 bits por cada carácter. En este caso, los 8 bits de menos peso se cargan en el registro TXREG mientras que el noveno (bit 8) se carga en el bit TX9D del registro TXSTA. Esta característica se habilita mediante el bit TX9 del mismo registro.

El módulo transmisor se habilita mediante el bit TXEN. En ese momento interviene el circuito generador de baudios, que determina la frecuencia o ritmo con el que se transmite cada bit. La frecuencia del circuito generador de baudios, que determinará la velocidad de comunicación, se establece mediante el registro SPBRG asociado al generador de baudios, una serie de bits de control y el oscilador principal del sistema (Fosc). Por último, el bit SPEN habilita la puerta serie y configura el terminal RC6/TX/CK, que puede actuar como línea TX de transmisión de datos en el modo asíncrono o como línea CK de reloj en el modo síncrono.

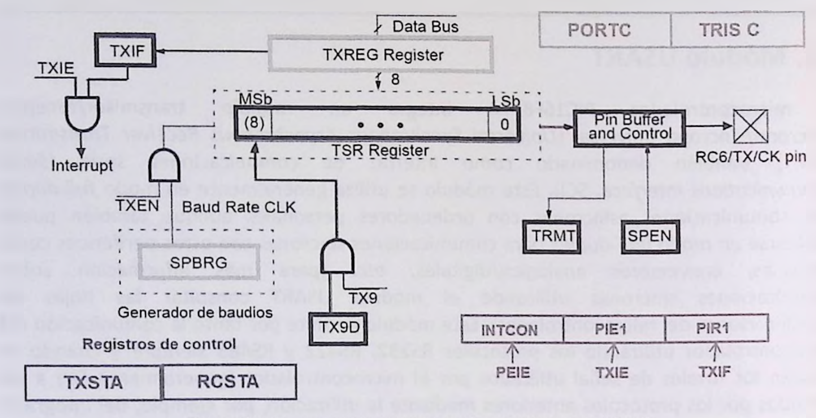


FIGURA 6.1: ESQUEMA DE BLOQUES DEL MÓDULO TRANSMISOR USART.

El circuito de recepción cuenta con un registro de desplazamiento de entrada serie y salida paralelo llamado RSR (Receiver Shift Register) al que no se puede acceder directamente. El bit SPEN del registro RCSTA, se emplea para habilitar la puerta serie, y con ello configurar el terminal RC7/RX/DT como línea Rx de recepción asíncrona, o bien como línea DT de datos en el modo síncrono. Por este terminal se van recibiendo los bits, que se van almacenando en el registro RSR para posteriormente copiarse al registro de recepción RCREG una ver recibidos los caracteres de 8/9 bits. El RCREG funciona internamente como un buffer de entrada tipo FIFO (primero en entrar primero en salir) capaz de almacenar dos caracteres. Cuando el RSR recibe un carácter, se copia sobre el primer nivel de la FIFO, el propio RCREG. Al recibir un segundo carácter, sin leer el primero se almacena en el segundo nivel. Al leer RCREG el programa lee el primer carácter recibido y si hubiera un carácter sin leer pasaría automáticamente al RCREG. En el caso de recibir un tercer carácter sin leer los dos anteriores se genera un error de desbordamiento que se refleja en el bit OERR. Otro tipo de error que se puede producir en la recepción es el llamado "error de trama" que se ve reflejado en el bit FERR.

Siempre que en el registro RCREG haya un carácter disponible, el bit RCIF del registro PIR1 se pone a nivel '1'. En este momento, si el bit RCIE del registro PIE1 estuviera también a nivel '1', se generaría una interrupción por recepción de un carácter.

El sistema de recepción se habilita mediante el bit CREN. El circuito generador de baudios (el mismo utilizado en la sección de transmisión) determina la frecuencia o ritmo con la que se recibe cada bit.

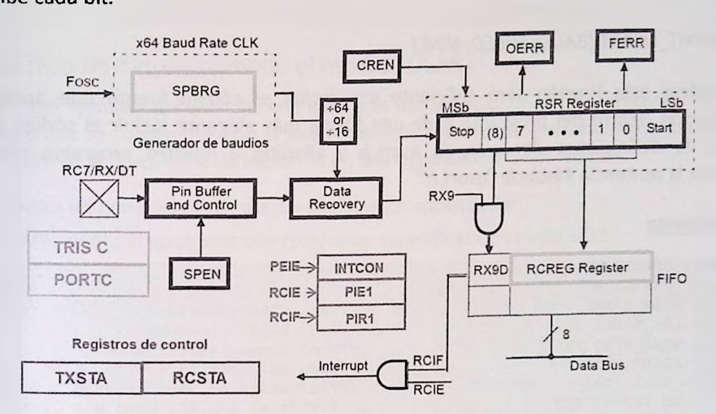


FIGURA 6.2: ESQUEMA DE BLOQUES DEL MÓDULO RECEPTOR USART.

6.1.1. Crea una macro que permita configurar del módulo USART en modo asíncrono

Solución:

En este ejemplo se va a configurar el módulo USART para comunicaciones en modo asíncrono de forma genérica. Para ello se creará una macro que permitirá introducir los valores necesarios en los registros de configuración.

Configuración

Para la creación de la macro será necesario definir diferentes constantes que determinen el modo de trabajo del módulo y que permitan ajustar los registros de configuración TXSTA, RCSTA y SPBRG. En este caso se definirá una máscara que ajuste el modo de trabajo en alta velocidad HIGH_SPEED = 0x04 y otra que ajuste el modo de funcionamiento en 9 bits NINE_BITS=0x40. Estas máscaras se utilizarán junto con la función OR (|) para ajustar los registros de configuración TXSTA y RCSTA. El registro SPBRG que determinará la velocidad del módulo obtiene su valor a partir del oscilador principal y el modo de funcionamiento a alta o baja velocidad a partir de las siguientes fórmulas:

$$\text{Baja velocidad} \rightarrow SPBRG = \frac{F_{osc}}{64(X+1)}$$

$$\text{Alta velocidad} \rightarrow SPBRG = \frac{F_{osc}}{16(X+1)}$$

Así pues, la macro para configurar el módulo USART en modo asíncrono tendrá la siguiente estructura:

#define INIT_USART (BAUD, SPEED, NINE)

Para utilizar esta función será suficiente con incluir el código fuente que aparece a continuación dentro del programa o de una forma más elegante incluir el código en un archivo de definiciones denominado usart.h y añadirlo a nuestro programa principal mediante la sentencia *#include "usart.h"*

Código fuente

```
#ifndef _SERIAL_H_
#define _SERIAL_H_
#define HIGH_SPEED 0x04                            //Modo alta velocidad
#define LOW_SPEED 0                                //Modo baja velocidad
#define NINE_BITS 0x40                             //Modo transmisión 9 bits
#define EIGHT_BITS 0                               //Modo transmisión 8 bits
#define _XTAL_FREQ      4000000                    //Frecuencia del oscilador principal
#define RX_PIN TRISC7                              //Define el pin RC7 como pin para
                                                   //recepción de datos
#define TX_PIN TRISC6                              //Define el pin RC6 como pin para
                                                   //transmisión de datos

/*MACRO PARA CONFIGURAR EL MÓDULO USART EN MODO ASÍNCRONO*/
//BAUD: Velocidad de comunicaciones (9600, 19200, 28800, etc.)
//SPEED: Modo alta velocidad para el generador de baudios => TRUE = On
//NINE: Utiliza transmission de 9 bits => FALSE=8bit
#define INIT_USART(BAUD,SPEED,NINE)\
        RX_PIN = 1;                                //Define los pines RC6 y RC7 para poder
        TX_PIN = 1;                                //ser utilizados por el módulo USART
        if (SPEED)SPBRG = ((int)(_XTAL_FREQ/((16UL * BAUD) +16)));  //Ajusta la
                                                                    //velocidad
        else SPBRG = ((int)(_XTAL_FREQ/((64UL * BAUD) +64)));       //del generador
                                                                    //de baudios
        RCSTA = (NINE|0x90);                       //Carga la configuración del módulo
                                                   //receptor
        TXSTA = (SPEED|NINE|0x20)                  //Carga la configuración del módulo
                                                   //transmisor

void putch(unsigned char);                         //Definición de la función putch
#endif
```

Simulación

Se puede probar el funcionamiento de la macro anterior llamando a la función INIT_USART(9600, HIGH_SPEED, EIGHT_BITS); desde el programa principal y observando el valor que toman los registros de configuración TXSTA, RCSTA y SPBRG.

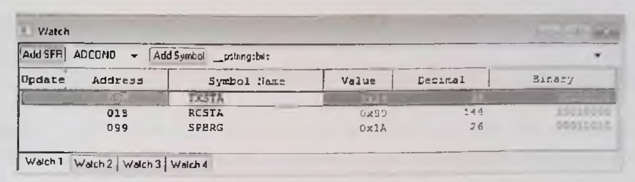


FIGURA 6.3: SIMULACIÓN DE LA CONFIGURACIÓN EL MÓDULO USART (WATCH).

6.1.2. Envío de datos utilizando el módulo USART

A partir del ejemplo anterior de configuración del módulo USART se pretende crear un programa que se comunique con el puerto serie del ordenador personal y escriba a través de un terminal la típica frase de iniciación en la programación "Hola Mundo!".

a) Indica las conexiones necesarias con el microcontrolador.
b) Implementa el programa que cumpla las especificaciones indicadas.

Solución:

a)

Consideraciones eléctricas

A la hora de conectar el microcontrolador PIC con el puerto serie del ordenador es importante tener en cuenta los niveles de tensión con los que se está trabajando. El puerto serie del ordenador sigue el estándar RS232 y los niveles de tensión con los que trabaja van de entre +3 y +15 V para el '1' lógico y de −3 a −15 V para el '0' lógico mientras que el microcontrolador trabaja con niveles de tensión entre 0 V y 3,3-5 V típicamente por lo que es necesario utilizar un convertidor de tensión del tipo MAX232 o similar para adaptar los niveles de señal. El esquema de la figura inferior muestra las conexiones necesarias entre el microcontrolador y el ordenador utilizando el integrado MAX232.

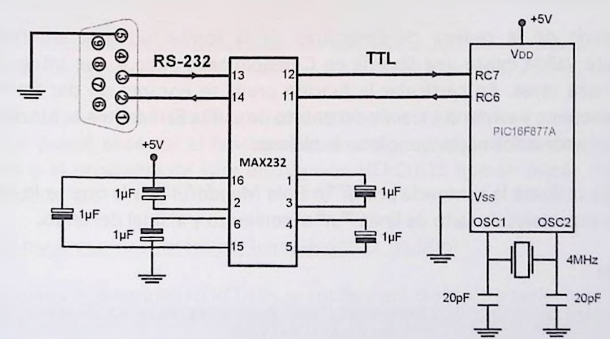


FIGURA 6.4: ESQUEMA DE CONEXIONES CON EL MICROCONTROLADOR PARA UTILIZAR EL MÓDULO DE COMUNICACIONES USART PROPUESTO EN EL EJERCICIO 6.1.2.

b)

Configuración

Para la configuración del módulo USART se utilizará la función INIT_USART detallada en el ejemplo anterior y cuyo código se incluirá dentro de la librería usart.h a la que se añadirá la definición de una nueva función que se encargará de enviar un byte de datos a través del módulo USART. Esta nueva función denominada *void putch(unsigned char byte)* tendrá como argumento un dato de tamaño byte y se encargará de transmitir el dato escribiéndolo en el registro TXREG cuando el módulo esté disponible (bit TXIF='1').

El código fuente de la función *putch* se incluirá dentro de la librería usart.c y tendrá la siguiente estructura:

```
void putch(unsigned char byte){

      while(!TXIF)                    //Se pone a uno cuando el modulo está libre
        continue;
      TXREG = byte;
}
```

El envío de datos utilizando la función putch se realiza byte a byte por lo que para enviar una cadena de caracteres sería necesario llamar a la función *putch* de forma consecutiva por cada carácter a enviar utilizando un código similar al inferior:

```
void main(void){
      const unsigned char texto[11]="Hola Mundo!";
      unsigned char i=0;
      INIT_USART(9600,HIGH_SPEED,EIGHT_BITS); //Configura el modulo USART
      while(1){
        for(i=0;i<11;i++){                    //Envía la cadena de caracteres
        putch(texto[i]);
        }
      }
}
```

Aunque el envío de la cadena de caracteres de la forma indicada anteriormente es completamente válida existe una librería en C denominada stdio.h que integra funciones que facilitan esta tarea. En particular la función printf se encarga de dar formato a una cadena de caracteres y enviarla a través del puerto de salida llamando a la función putch las veces que sean necesarias hasta completar la cadena.

En este caso se utilizará la sentencia *printf("\n Hola Mundo!\n");* a la que se le han añadido los caracteres especiales de salto de línea "\n" al comienzo y al final del texto.

Código fuente

```
#include <stdio.h>           //Librería con funciones para el control de
                             //entradas y salidas
#include <htc.h>             //Librería que incluye las definiciones del
                             //microcontrolador
```

```
#include "usart.h"                          //Librería para el módulo de
                                            //comunicaciones USART

#define _XTAL_FREQ 4000000                  //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

void main(void){

      INIT_USART(9600,HIGH_SPEED,EIGHT_BITS);      //Configura el modulo USART
      while(1){
        printf("\n Hola Mundo!\n");                //Escritura utilizando la
                                                   //función printf
      }
}
```

Simulación

Se puede comprobar el funcionamiento del programa mediante el simulador MPLAB SIM habilitando la visualización del módulo de comunicaciones UART a través de la ventana OUTPUT (menú *debugger/settings*) como se muestra en la figura inferior.

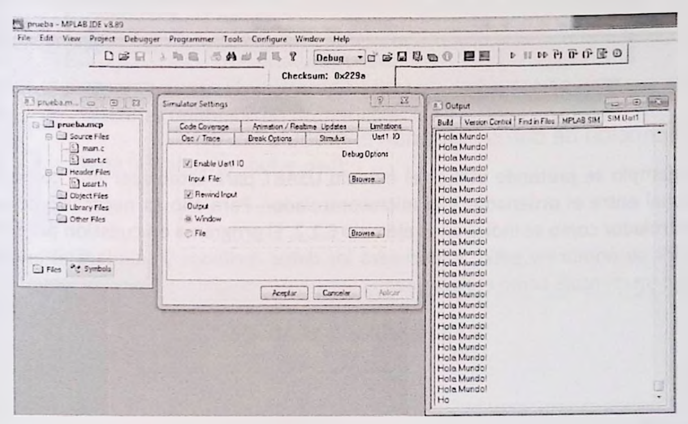


FIGURA 6.5: CONFIGURACIÓN DE MPLAB SIM PARA UTILIZAR EL MÓDULO USART Y SIMULACIÓN DEL EJERCICIO 6.1.2.

En caso de disponer del *hardware* necesario para conectar el microcontrolador al ordenador se puede observar el funcionamiento del programa utilizando el hyperterminal de Windows o el programa de libre distribución HERCULES que se puede descargar de la siguiente dirección:

http://www.hw-group.com/products/hercules/index_es.html

Una vez instalado el programa HERCULES se configurará del puerto serie de comunicaciones para que coincida con la configuración del módulo USART (9600 baudios, 8 bits y sin paridad) y se abrirá el puerto de comunicaciones. Al ejecutar el programa se observa su funcionamiento en la ventana de HERCULES como se muestra en la figura inferior.

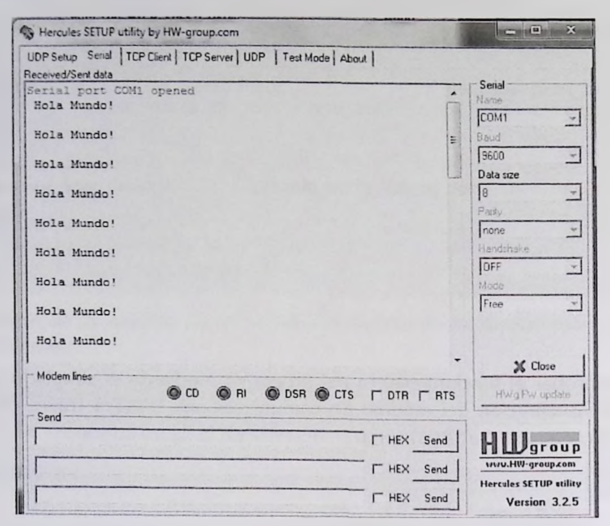


FIGURA 6.6: SIMULACIÓN DEL EJERCICIO 6.1.2 UTILIZANDO EL PROGRMA HERCULES.

6.1.3. Recepción de datos utilizando el módulo USART

En este ejemplo se pretende utilizar el módulo USART para establecer una comunicación bidireccional entre el ordenador y el microcontrolador. Para ello es necesario conectar el microcontrolador como se indica en el ejemplo 6.1.2. El programa en cuestión preguntará al usuario por su nombre y edad. Almacenará los datos recibidos y saludará al interlocutor mostrando un mensaje como el siguiente:

"Hola NOMBRE de EDAD años"

Solución:

Configuración

Para establecer la comunicación entre el microcontrolador y el ordenador se utilizará el módulo USART del microcontrolador configurado en modo 8 bits a 9600 baudios.

Para recibir los caracteres del módulo USART se creará una nueva función *getche* que se encargará de recoger el dato recibido del registro RCREG. La función tendrá la siguiente estructura:

```
unsigned char getche(void)
```

La definición de esta nueva función se deberá incluir en el archivo usart.h y el código que se muestra a continuación se incluirá en el archivo usart.c junto con la función *putch*.

```
unsigned char getche() {

      while(!RCIF)           //Se pone a '1' cuando se ha recibido un dato
         continue;
      return RCREG;
}
```

Como la función getche recibe los datos byte a byte es necesario llamar a la función las veces necesarias hasta recoger todos los caracteres correspondientes tanto al nombre como a la edad hasta encontrar el carácter especial de salto de línea (LF) o retorno de carro (CR). De esto se encargará la función *gets()* incluida en la librería stdio.h, al igual que lo hacía la función *printf* con la función *putch.* La función *gets()* recibe como parámetro un puntero a una cadena de caracteres donde almacenará los datos recibidos a través del teclado hasta encontrar los caracteres especiales LF o CR.

Al recibir el dato de la edad en forma de cadena de caracteres puede ser necesario para trabajar con este valor convertirlo a un número entero para lo que se puede utilizar en este caso la función *atoi()* incluida en la librería stdlib.h que convierte el valor ascii a entero.

```
e_dad=atoi(edad);
```

Por último, el programa mostrará un mensaje utilizando la función *fprintf()* que construirá una cadena de caracteres utilizando las dos variables buf y e_dad.

printf("\n Hola %s de %3i años.\n",buf,e_dad);

<u>**Código fuente**</u>

```
#include <htc.h>
#include <stdio.h>
#include <stdlib.h>                //Librería con funciones adicionales atoi()
#include "usart.h"

#define _XTAL_FREQ 4000000 //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

void main(void){
      unsigned char buf[80],edad[3],e_dad;
      di();                                   //Desabilita interrupciones

      INIT_USART(9600,HIGH_SPEED,EIGHT_BITS); //Configura el modulo USART

      while(1){
         printf("\n Hola! ¿Como te LLamas?"); //Mensaje de salida
         gets(buf);                           //Almacena la respuesta del
                                              //usuario en buf
         printf("\n ¿Cuantos años tienes?");  //Mensaje de salida
         gets(edad);                          //Almacena la respuesta del
                                              //usuario en edad
         e_dad=atoi(edad);                    //Convierte la edad a número
                                              //entero
         printf("\n Hola %s de %3i años.\n",buf,e_dad);    //Respuesta
      }
}
```

Simulación

Como se indicó en el ejemplo anterior se utilizará el programa HERCULES para comprobar el funcionamiento del programa. El funcionamiento se muestra en la siguiente figura:

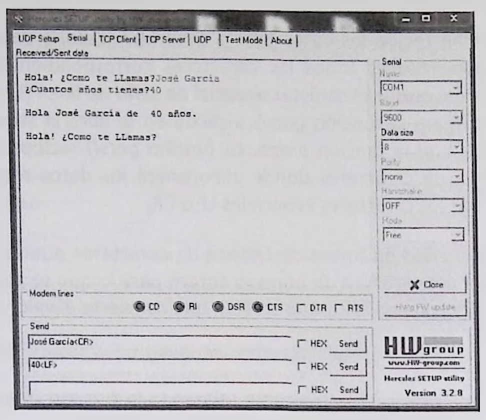


FIGURA 6.7: SIMULACIÓN EL EJERICIO 6.1.3 UTILIZANDO EL PROGRAMA HERCULES.

6.1.4. Adivina el número

En este ejemplo se pretende realizar un programa que genere un número aleatorio entre 0 y 99 y le rete al usuario a adivinar de qué número se trata. En caso de no acertar el número el programa indicará si el número en cuestión es menor o mayor que el introducido. Cuando se acierte el número el programa indicará el número de intentos realizados por el usuario y le preguntará si desea volver a jugar.

Solución:

Configuración

El programa consistirá en utilizar el puerto de comunicaciones serie USART para comunicarse con el usuario mostrando mensajes de texto y recogiendo los valores introducidos. Para ello será necesario configurar el módulo USART.

```
INIT_USART(9600,HIGH_SPEED,EIGHT_BITS);
```

Para mostrar los mensajes se utilizará la función *printf()* y para guardad los datos recibidos la función *gets()* combinada con la función *atoi()* que convierte de ascii a entero tal y como se ha mostrado en el ejemplo anterior.

Además, será necesario utilizar la función *rand()* incluida en la librería stdlib.h que permite generar un número aleatorio entre 0 y 32767. Esta función se combina junto con la función

srand() también de la librería stdlib.h que inicializa el generador de número aleatorios a partir de un número o semilla. Para evitar que cada vez que se inicie el programa por primera vez se repitan la secuencia de números se utilizará el temporizador TIMER0 como semilla, tomando el valor de TMR0 en el momento en el que se comienza el juego.

Código fuente

```
#include <htc.h>
#include <stdio.h>
#include <stdlib.h>          //Librería con funciones adicionales atoi()
#include "usart.h"

#define _XTAL_FREQ 4000000                          //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

void main(void){
      unsigned char valor, num[10],numero,i,acierto;      //definición de
                                                          //variables
      di();                                   //desabilita interrupciones
      TOCS=0;                                 //arranca el timer0
      INIT_USART(9600,HIGH_SPEED,EIGHT_BITS); //Configura el modulo USART -
                                              //Preferencias definidas en usart.h
      while(1){
        printf("\n Adivina un numero del 0 al 99! ");     //Mensaje de salida

        printf("\n Quieres jugar? [S/N]: ");  //Mensaje de salida
        valor=getche();                       //Lee la respuesta del usuario
        if(valor=='S'){
                i=0;                          //Inicia el contador de intentos
                acierto=0;                    //desactiva el indicador de acierto
                srand((int)TMR0);             //inicializa el generador de n°s
                                              //aleat.
                valor=rand()%100;             //obtiene un n° aleatorio entre 0
                                              //y 99
                while(!acierto){
                 i++;
                 printf("\n Introduce un numero del 0 al 99: ");  //Mensaje de
                                                                   //salida
                 gets(num);                   //Lee la respuesta del usuario
                 numero=atoi(num);            //Convierte el valor leido a
                                              //número entero
                 if(valor==numero){
                      acierto=1;              //activa el indicador de acierto
                      printf("\n ENHORABUENA!!! Has acertado en %3i
                                                             intentos.\n",i);
                 }
                 else if(valor<numero){
                      printf("\n El numero %3i es mayor que el numero
                                                       buscado.\n",numero);
                 }
                 else{
                      printf("\n El numero %3i es menor que el numero
                                                       buscado.\n",numero);
                 }
                }
        }
      }
}
```

Simulación

Como se indicó en el ejemplo anterior se utilizará el programa HERCULES para comprobar el funcionamiento del programa conectado al puerto serie del ordenador. El funcionamiento se muestra en la siguiente figura:

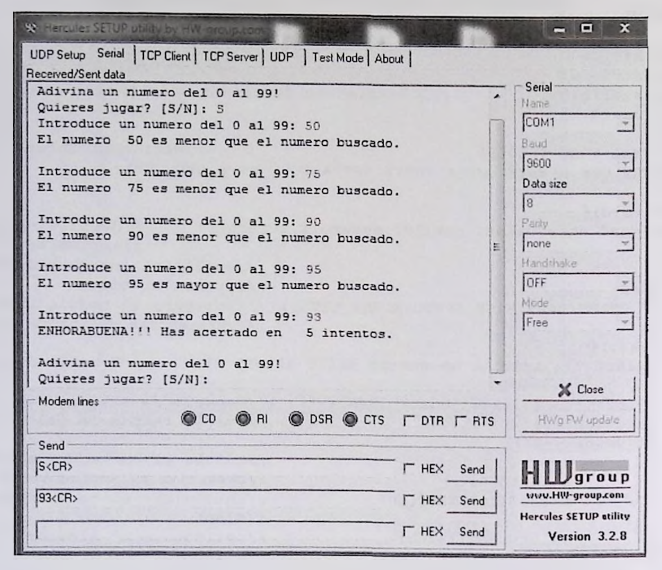


FIGURA 6.8: SIMULACIÓN DEL EJERCICIO 6.1.4 UTILIZANDO EL PROGRAMA HERCULES.

Realiza un programa que utilice el módulo USART para implementar un juego en el que el usuario tenga que adivinar una palabra. El usuario irá introduciendo caracteres en mayúsculas y el programa le indicará si la palabra contiene ese carácter y la posición o posiciones que ocupa en la palabra.

6.2. Módulo MSSP (Master Synchronous Serial Port)

El modulo MSSP es un interfaz de comunicaciones serie muy utilizado para comunicar el microcontrolador con dispositivos periféricos o con otros microcontroladores. Entre los dispositivos que utilizan habitualmente este módulo de comunicaciones se pueden encontrar memorias EEPROM, registros de desplazamiento, controladores de displays, conversores analógico/digitales, etc.

El módulo MSSP tiene asociados tres registros. El registro de status SSPSTAT (A2.17) y dos registros de control SSPCON (A2.18) y SSPCON2 (A2.19) que determinarán el

funcionamiento del mismo en modo I²C o SPI tanto funcionando en modo maestro como en modo esclavo.

Los terminales del microcontrolador asociados al módulo MSSP son los siguientes:

TERMINAL	SPI	I²C
RC3	Serial Clock (SCK)	Serial Clock (SCL)
RC4	Serial Data In (SDI)	Serial Data (SDA)
RC5	Serial Data Out (SDO)	–
RA5	Slave select ($\overline{SS}$)	–

Registros de control

SSPCON	SSPCON2	SSPSTAT

MODO SPI

MODO I²C

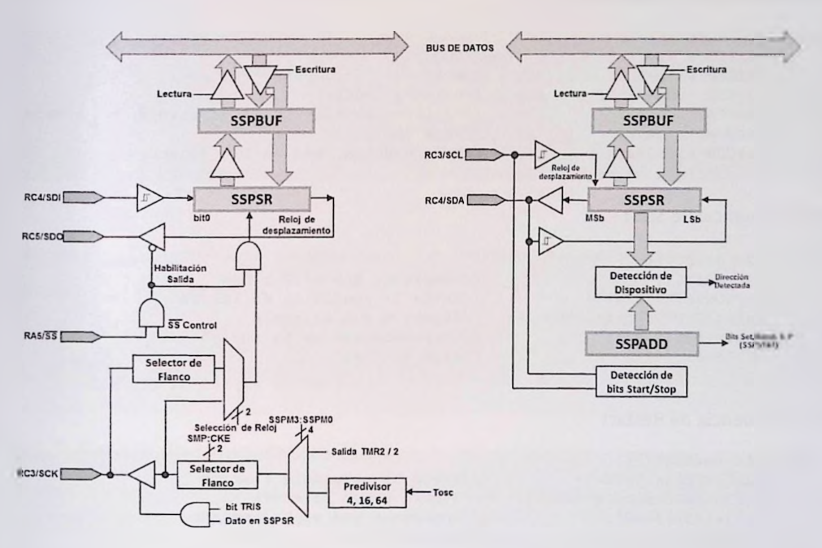


FIGURA 6.9: ESQUEMA DE BLOQUES DEL MÓDULO MSSP.

6.2.1. Configuración del módulo MSSP en modo I²C maestro

Configura el módulo I²C del microcontrolador PIC16F877A para trabajar en modo maestro.

Solución:

Para configurar el módulo será necesario ajustar correctamente los registros de configuración SSPSTAT (A2.17), SSPCON (A2.18) y SSPCON2 (A2.19). Para ello se utilizará la

función *i2c_init()*. Además, se definirán diferentes funciones adicionales que permitirán interactuar de forma sencilla con el módulo tales como *i2c_start()*, *i2c_restart()*, *i2c_stop()*, *i2c_idle()*, *i2c_read()*, *i2c_write()* y *i2c_sendnack()* cuya definición y código se introducirán en los archivos "i2c.h" e "i2c.c" respectivamente.

Configuración

La función de configuración deberá configurar los pines SDA y SCL para su control a través del módulo MSSP (TRISC4 = 1; TRISC3 = 1;), ajustar el registro SSPCON (A2.18) para encender el módulo y establecer modo maestro (SSPCON = 0x28), ajustar la velocidad del reloj (SSPADD = 0x0A), bajar los flags (SSPCON2 = 0;) y habilitar el control de velocidad (SSPSTAT=0x80;).

Código fuente

//Configuración del módulo I2C

```
void  i2c_init(){
      TRISC3 = 1;                     //SLC
      TRISC4 = 1;                     //SDA
      SSPADD = 0x0A;                  //Reloj a 100KHz
      SSPSTAT = 0x80;                 //Habilita el control de velocidad del módulo
      SSPCON2 = 0x00;                 //Baja los flags
      SSPCON = 0x28;                  //Modo maestro, módulo I2C encendido
}
```

Envía secuencia de Start

```
void  i2c_start(){
      i2c_idle();                     //Comprueba que está libre
      SSPCON2bits.SEN=1;              //Envía la condición de inicio
      while(SSPCON2bits.SEN==1);      //Espera a que se envíe
      while(SSPIF==0);                //Comprueba que se ha enviado
      SSPIF=0;                        //Baja el flag
}
```

Envía secuencia de Restart

```
void  i2c_restart(){
      SSPCON2bits.RSEN=1;             //Envía la secuencia restart
      while(SSPCON2bits.RSEN);        //Espera a que se envíe
      while(SSPIF==0);                //Comprueba que se ha enviado
      SSPIF=0;                        //Baja el flag
}
```

Envía Secuencia de Stop

```
void  i2c_stop(){
      SSPCON2bits.PEN=1;              //Envía la secuencia de Stop
      while(SSPCON2bits.PEN);         //Espera a que se envíe
      while(SSPIF==0);                //Comprueba que se ha enviado
      SSPIF=0;                        //Baja el flag
}
```

Comprueba si el módulo está ocupado

```
void  i2c_idle(){
while( ( SSPCON2 & 0x1F ) | SSPSTATbits.R_nW );      //Comprueba que el módulo
                                                      //está libre
}
```

Lee un byte

```
unsigned char    i2c_read(){
      i2c_idle();
      SSPCON2bits.RCEN=1;                    //Activa el modo recepción
      while(SSPCON2bits.RCEN==1);            //Espera a recibir el byte
      while(SSPSTATbits.BF==0);              //Espera a que el buffer esté lleno
      return SSPBUF;                         //Devuelve el dato
}
```

Envía un byte – Devuelve 0 en caso de error

```
unsigned char    i2c_write(unsigned char byte){
      i2c_idle();
      SSPBUF=byte;
      while(SSPSTAT&0x05);             //Comprueba si hay transmisión en curso R/W y
                                       //si el buffer está vacío BF
      if(SSPCON2bits.ACKSTAT==1){            //No ha recibido confirmación
         SSPCON2bits.PEN=1;                  //Envía secuencia de Stop
         return  0 ;                         //devuelve 0
      }
      else{                                  //Ha recibido confirmación
         while(SSPIF==0);                    //Comprueba que ha transmitido
         SSPIF=0;                            //Baja el flag
         return 1;                           //Devuelve 1
      }
}
```

Envía secuencia NACK

```
void  i2c_sendnack(){
      SSPCON2bits.ACKDT=1;
      SSPCON2bits.ACKEN=1;
      while(SSPCON2bits.ACKEN==1);
}
```

6.2.2. Sensor de temperatura TC74

Realiza un programa que utilice el sensor digital de temperatura TC74 (DS21462D) para mostrar la temperatura en la pantalla LCD HITACHI HD44780.

a) Indica las conexiones necesarias con el microcontrolador.
b) Implementa el programa que cumpla las especificaciones indicadas.

<u>Solución:</u>

a) Para el funcionamiento del microcontrolador será necesario conectar correctamente las entradas VDD y VSS a +5 V y 0 V respectivamente y el oscilador, en este caso un oscilador de cristal a 4 MHz, a las entradas OSC1 y OSC2 tal y como aparece en la figura inferior. El sensor de temperatura TC74 se conectará siguiendo las indicaciones de la hoja de

especificaciones con sus salidas SDA y SCL conectadas a los terminales de comunicaciones I²C del microcontrolador RC4 y RC3 respectivamente. Se utilizarán además dos resistencias de pull-up conectadas a las líneas de comunicaciones I²C que mantendrán estas líneas en alto '5 V' cuando estén sin utilizar. La conexión de la pantalla LCD se realizará siguiendo las instrucciones del fabricante para utilizar el modo de comunicaciones de 4 bits a través de los terminales RD0-RD4.

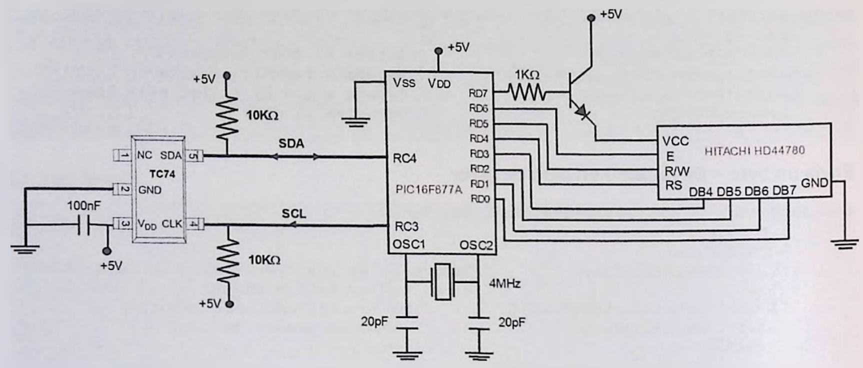


FIGURA 6.10: ESQUEMA DE CONEXIONES CON EL MICROCONTROLADOR PARA UTILIZAR EL SENSOR DE TEMPERATURA DIGITAL TC74 PROPUESTO EN EL EJERCICIO 6.2.2.

b) Para la comunicación del sensor de temperatura digital será necesario configurar el módulo de comunicaciones I²C del microcontrolador en modo maestro como se mostró en el ejemplo anterior. Una vez configurado el módulo se deberá establecer la comunicación con el dispositivo siguiendo las indicaciones de la hoja de especificaciones del sensor de temperatura TC74.

Configuración

Para leer el valor de la temperatura del módulo TC74 es necesario seguir las instrucciones del fabricante. El protocolo a seguir para leer la temperatura se detalla en la figura inferior.

Read Byte Format

S	Address	WR	ACK	Command	ACK	S	Address	RD	ACK	Data	NACK	P
	7 Bits			8 Bits			7 Bits			8 Bits		

Slave Address

Command Byte: selects which register you are reading from.

Slave Address: repeated due to change in data-flow direction.

Data Byte: reads from the register set by the command byte

FIGURA 6.11: ESQUEMA PARA LA LECTURA DE DATOS UTILIZANDO EL SENSOR DE TEMPERATURA DIGITAL TC74.

Para leer la temperatura se creará una función *TC74_i2c_readbyte()* que se encargue de la comunicación. Esta función hará uso de la librería "i2c.h" donde se encuentran definidas todas las funciones de comunicaciones para trabajar con el módulo I²C (véase ejemplo anterior). Así, la función tendrá la siguiente estructura:

```
#define WRITE 0
#define READ 1

signed char TC74_i2c_readbyte(unsigned char add, unsigned char dato){
      signed char temp=0;
      i2c_start();                    //Envía la secuencia de Start (S)
      if(i2c_write(add+WRITE)==0){    //Envía la dir. del módulo TC74
                                      //en modo escritura
        return 0;                     //Retorna 0 en caso de error
      }
      If(i2c_write(dato)==0){         //Envía el registro que se va
                                      //a leer (dato)
      return 0;                       //Retorna 0 en caso de error
      }
      i2c_restart();                  //Envía la secuencia de Restart
      if(i2c_write(add+READ)==0){     //Envía la dirección del módulo TC74
                                      //en modo lectura
      return 0;                       //Devuelve 0 en caso de error
      }
      temp=i2c_read();                //Guarda el valor de recibido
      i2c_sendnack();                 //Envía secuencia NACK
      i2c_stop();                     //Envía secuencia de Stop
      return temp;                    //Devuelve el valor obtenido
}
```

Una vez obtenido el valor de la temperatura se deberá representar en la pantalla LCD. Para ello se hará uso de la función printf incluida en la librería "stdio.h" como ya se comentó en ejemplos anteriores.

Código fuente

```
#include <htc.h>
#include "i2c.h"
#include "lcd.h"
#include "stdio.h"
#include "TC74.h"
#define _XTAL_FREQ 4000000                  //Oscilador Interno de 4MHZ
#define TC74_ADDRESS 0x9A                   //Dirección del módulo TC74
#define TC74_TEMP_REG 0x00                  //Dirección del registro de
                                            //temperatura
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

void putch(unsigned char x);                //Definición de la función putch

void main(void){                            //Función principal
      signed char temp;                     //Variable para almacenar la
                                            //temperatura
      di();                                 //Deshabilita interrupciones
      lcd_init();                           //Inicia la pantalla LCD
      i2c_init();                           //Inicia el módulo I2C
      while(1){                             //Bucle infinito
        temp=TC74_i2c_readbyte(TC74_ADDRESS, TC74_TEMP_REG); //Lee la temperatura
                                                             //del TC74
        if(temp!=0){             //Comprueba que se haya leído correctamente
              lcd_goto(0);       //Posiciona el cursor al comienzo de la LCD
              printf("Temp = %+d C  ",temp);     //Muestra la temperatura
              __delay_ms(100);         //Espera 100ms para volver a leer Temp
        }
      }
}
```

```
void putch(unsigned char x){ lcd_putch(x); }
```

6.2.3. Memoria EEPROM 24LC256

Realiza un programa basado en un microcontrolador PIC trabajando a 4 MHz que almacene en la posición de memoria 0x0000 de una memoria EEPROM externa 24LC256 (DS21203M) el número de veces que se ha arrancado el programa almacenado en el PIC (se supondrá un estado inicial de los registros de memoria 0XFF).

a) Indica las conexiones necesarias con el microcontrolador.
b) Implementa el programa que cumpla las especificaciones indicadas.

Solución:

a) Para el funcionamiento del microcontrolador será necesario conectar correctamente las entradas VDD y VSS a +5 V y 0 V respectivamente y el oscilador, en este caso un oscilador de cristal a 4 MHz, a las entradas OSC1 y OSC2 tal y como aparece en la figura inferior. La memoria 24LC256 se conectará siguiendo las indicaciones de la hoja de especificaciones con sus salidas SDA y SCL conectadas a los terminales de comunicaciones I^2C del microcontrolador RC4 y RC3 respectivamente. Se utilizarán además dos resistencias de pull-up conectadas a las líneas de comunicaciones I^2C que mantendrán estas líneas en alto '5 V' cuando estén sin utilizar. Los terminales A0-A2 que determinan los tres últimos bites de la dirección se conectarán a tierra '0 V' determinando una dirección del dispositivo 0b0101000. El terminal WP (*write-protect*) se conectará también a tierra '0 V'.

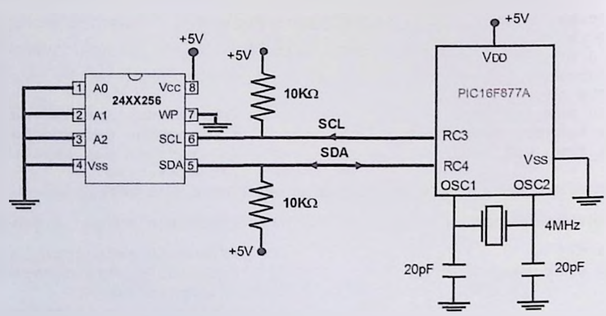


FIGURA 6.12: ESQUEMA DE CONEXIONES CON EL MICROCONTROLADOR PARA UTILIZAR LA MEMORIA EEPROM 24LC256 PROPUESTO EN EL EJERCICIO 6.2.3.

b) Para la comunicación de la memoria EEPROM será necesario configurar el módulo de comunicaciones I^2C del microcontrolador en modo maestro como se mostró en el ejemplo 6.2.1. Una vez configurado el módulo se deberá establecer la comunicación con el dispositivo siguiendo las indicaciones de la hoja de especificaciones para las operaciones de lectura y escritura que se detallan en la figura inferior.

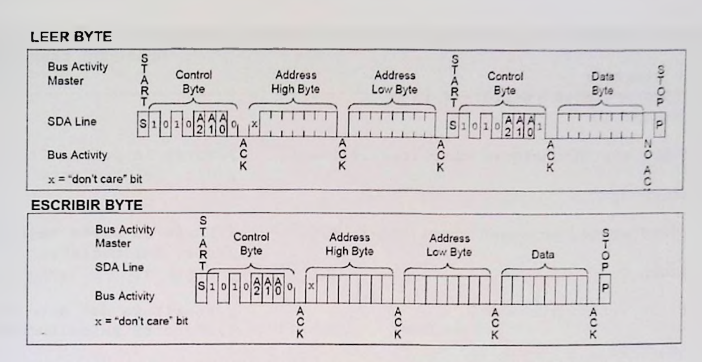


FIGURA 6.13: ESQUEMA PARA LA LECTURA Y ESCRITURA DE DATOS EN LA MEMORIA EEPROM 24LC256.

Configuración

Para realizar la comunicación con la memoria EEPROM se crearán dos funciones EEPROM_*i2c_readbyte()* y EEPROM_*i2c_writebyte()*, que se encargarán de leer y escribir en la memoria EEPROM respectivamente. Estas funciones harán uso de la librería "i2c.h" donde se encuentran definidas todas las funciones de comunicaciones para trabajar con el módulo I²C (véase ejemplo 6.2.1). Así, las funciones tendrán la siguiente estructura:

```
//Lee un byte del dispositivo I2C esclavo de DIRECCION=add en su REGISTRO=reg
unsigned char EEPROM_i2c_readbyte(unsigned char add, unsigned int reg){
      unsigned char temp=0;
      i2c_start();                      //Envío de la secuencia de Start
      if(i2c_write(add+WRITE)==0){      //Envío de la dir. i2c de la EEPROM en
                                        //modo escritura
        return 0;
      }
      if(i2c_write((unsigned char)(reg>>8))==0){     //Envía la parte alta de la
                                                     //dir. del registro
      return 0;
      }
      if(i2c_write((unsigned char)(reg))==0){        //Envía la parte baja de la
                                                     //dir. del registro
      return 0;
      }
      i2c_restart();                    //Envío de la secuencia de Restart
      if(i2c_write(add+READ)==0){       //Envío de la dirección i2c de la EEPROM
                                        //en modo lectura
      return 0;
      }
      temp=i2c_read();                  //Lectura del dato seleccionado
      i2c_sendnack();                   //Envío de la secuencia NACK
      i2c_stop();                       //Envío de la secuencia de Stop
      return temp;                      //Retorno del dato
}

//Escribe el BYTE=dato en el dispositivo I2C esclavo de DIRECCION=ADD en su
//REGISTRO=reg
```

```
unsigned char EEPROM_i2c_writebyte(unsigned char add, unsigned int reg,
                                                      unsigned char dato){
      i2c_start();
      if(i2c_write(add+WRITE)==0){
        return 0;
      }
      if(i2c_write((unsigned char)(reg>>8))==0){     //Envía la parte alta de la
                                                     //dir. del registro
      return 0;
      }
      if(i2c_write((unsigned char)(reg))==0){        //Envía la parte baja de la
                                                     //dir. del registro
      return 0;
      }
      if(i2c_write(dato)==0){                        //Escritura del dato en el
                                                     //registro seleccionado
      return 0;
      }
      i2c_stop();                      //Envío de la secuencia de Stop
      return 1;
}
```

Código fuente

```
#include <htc.h>
#include "i2c.h"               //Librería con las rutinas asociadas al módulo i2c
#include "24LC256.h"           //Librería con las rutinas asociadas a la memoria
                               //EEPROM

#define _XTAL_FREQ 4000000          //Oscilador Interno de 4MHZ
#define EEPROM_ADDRESS 0xA0         //Dirección del módulo EEPROM (véase
                                    //especificaciones)
#define EEPROM_REG 0x0000           //Dirección del registro de memoria (véase
                                    //especificaciones)

__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);
void putch(unsigned char x);

void main(void){
      unsigned char dato,i;         //Definición de variables de programa
      di();                         //Se deshabilitan las interrupciones
      i2c_init();
      dato=EEPROM_i2c_readbyte(EEPROM_ADDRESS, EEPROM_REG);  //Lectura del valor
                                                             //almacenado
      dato++;                             //Se incrementa el valor en 1
      i=EEPROM_i2c_writebyte(EEPROM ADDRESS, EEPROM_REG, dato); //Se escribe el
                                                                //nuevo valor
      while(1);                           //Fin del programa
}
```

Realiza un programa que permita borrar todos los registros de la memoria EEPROM (ponerlos a 0xFF).

Realiza un programa que permita ir almacenando los valores de temperatura mostrados en el ejercicio 6.2.2 en la memoria EEPROM cada hora y permita mostrarlos en la pantalla LCD al accionar un pulsador.

7. Módulos periféricos

7.1. Memoria interna EEPROM

El microcontrolador PIC16F877A cuenta con una memoria EEPROM interna de 256 bytes. La memoria EEPROM no es accesible directamente a través de las direcciones de memoria del microcontrolador sino que se accede a ella a través de varios registros especiales de memoria EEADR y EEDAT utilizando los registros de control EECON1 y EECON2.

La utilización de esta memoria se puede realizar de forma sencilla mediante la utilización de macros o funciones pre-programadas incluidas en la librería "htc.h". Las macros y funciones que se utilizan para leer/escribir en la EEPROM son las siguientes:

MACROS

Inicia los datos en memoria EEPROM en bloques de 8 bytes comenzando en la dirección 0x00 e incrementando la dirección en 8 posiciones cada vez que se llama. Solo se puede utilizar al comienzo del programa y sirve para inicializar la memoria EEPROM en el momento de la programación del microcontrolador.

```
__EEPROM_DATA(a,b,c,d,e,f,g,h)
```

Escribe el dato de tamaño byte almacenado en value en la posición de memoria EEPROM indicada por addr.

```
#define EEPROM_WRITE(addr, value) \
do{ \
        while(WR)continue;EEADR=(addr);EEDATA=(value); \
        CARRY=0;if(GIE)CARRY=1;GIE=0; \
        WREN=1;EECON2=0x55;EECON2=0xAA;WR=1;WREN=0; \
        if(CARRY)GIE=1; \
}while(0)
```

Lee el byte almacenado en la dirección addr de memoria EEPROM y lo almacena en EEDATA.

```
#define  EEPROM_READ(addr) ((EEADR=(addr)),(RD=1),EEDATA)
```

FUNCIONES

Escribe el dato de tamaño byte almacenado en value en la posición de memoria EEPROM indicada por addr.

```
eeprom_write(unsigned char addr, unsigned char value){
      EEPROM_WRITE(addr, value);
}
```

Devuelve el dato de tamaño byte almacenado en la posición de memoria EEPROM indicada por addr.

```
unsigned char eeprom_read(unsigned char addr){
      while (WR) continue;
      return EEPROM_READ(addr);
}
```

7.1.1. Almacenamiento de datos no volátiles en memoria interna EEPROM

Se pretende crear un programa para un microcontrolador PIC16F877A que lleve la cuenta del número de veces que se ha reiniciado el programa en el registro de dirección 0x00 de memoria EEPROM interna.

Solución:

Para la realización del programa será necesario iniciar el registro 0x00 de la EEPROM interna en el momento de programación asignándole el valor 0x00 utilizando la siguiente macro:

```
__EEPROM_DATA(0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF);     //Inicializa los 8 primeros
                                                            //bytes de la EEPROM interna
```

A continuación, en el código del programa, se deberá leer el primer registro, incrementarlo y almacenarlo de nuevo en la misma posición de memoria.

Código fuente

```
#include<htc.h> //Incluimos libreria del micro a usar
__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);
#define _XTAL_FREQ 4000000                    //Oscilador Interno de 4MHZ

unsigned char s[10];

void main(void){
      unsigned char data;
      __EEPROM_DATA(0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF);  //Inicializa el
                                                               //contador a '0'
      di();                                   //Se deshabilitan interrupciones
      data=eeprom_read(0);                    //Se lee el contador
      data++;                                 //Se incrementa el contador
      eeprom_write(0,data);                   //Se almacena el contador
      NOP();                                  //Fin del programa
}
```

Simulación

Se ejecuta el programa utilizando el simulador MPLABSIM® a la vez que se visualizan los valores almacenados en los registros de memoria EEPROM interna (menú *View/EEPROM*).

Cada vez que hace un 'reset' y se vuelve a ejecutar el programa el valor almacenado en el registro de memoria 0x00 se incrementa en 1.

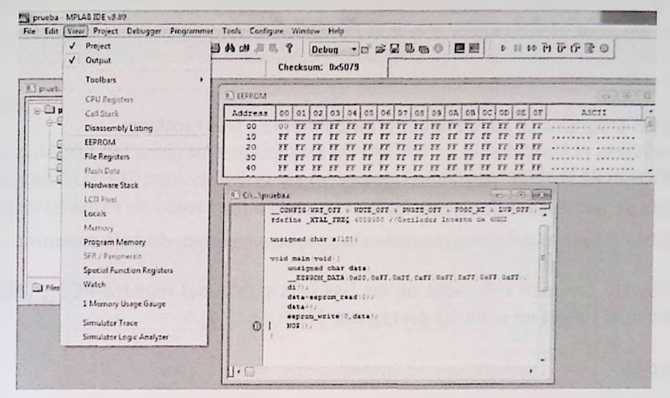


FIGURA 7.1: SIMULACIÓN DE LA ESCRITURA EN LA MEMORIA EEPROM INTERNA PROPUESTO EN EL EJERCICIO 7.1.1.

Realiza un programa que almacene los números del 0 al 255 en cada una de las posiciones de la memoria EEPROM.

7.2. Circuito de Reset

El microcontrolador PIC16F877A integra un circuito lógico conectado a la entrada de Reset que permite diferenciar entre varios motivos de Reset.

El estado de algunos registros no se modifica ante un Reset pero muchos otros ven alterado su valor ante un Reset (véase Anexo 2). En la figura inferior se muestra un diagrama simplificado del circuito de Reset.

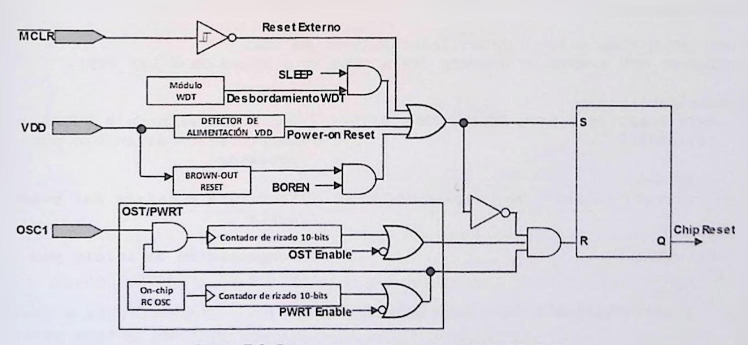


FIGURA 7.2: ESQUEMA DE BLOQUES DEL CIRCUITO DE RESET.

7.2.1. Comprobación del origen de reset al inicio del programa

Se propone crear un programa basado en un microcontrolador PIC16F877A trabajando a 4 MHZ que verifique el motivo por el que se ha reiniciado el programa.

Solución:

El reinicio del programa se puede deber principalmente a tres motivos:

1. Conexión de la alimentación.
2. Caída de tensión por debajo de los límites permitidos.
3. Desbordamiento del perro guardián sin encontrarse en modo bajo consumo.

Los motivos 1 y 2 vienen reflejados en los bits POR y BOR del registro PCON (A2.16) y el tercer motivo se refleja en el bit TO del registro STATUS (A2.1).

Configuración

Para verificar el origen del reset será necesario habilitar los motivos de reset en la palabra de configuración (A2.20) del microcontrolador utilizando la directiva __CONFIG de la siguiente manera:

```
__CONFIG(WRT_OFF & WDTE_ON & PWRTE_OFF & FOSC_XT & BOREN_ON & LVP_OFF);
```

La verificación del origen de reset se realizará comprobando el estado de los bits asociados a cada uno de los motivos anteriores y enviando a través del módulo USART un mensaje que indique dicho motivo. Para ello se empleará la librería utilizada en el ejemplo 6.2.1 "usart.h".

Código fuente

```
#include <stdio.h>
#include <htc.h>
#include "usart.h"

#define _XTAL_FREQ 4000000 //Oscilador Interno de 4MHZ
__CONFIG(WRT_OFF & WDTE_ON & PWRTE_OFF & FOSC_XT & BOREN_ON & LVP_OFF);

void main(void){
       INIT_USART(9600,HIGH_SPEED,EIGHT_BITS); //Configura el modulo USART
       if(!nPOR){                              //Comprobación de inicio por
                                               //conexión
          nPOR=1;
          printf("\n Reset por encendido!\n"); //Escritura a través del puerto
                                               //serie
       }
       if(!nBOR){                              //Comprobación de inicio por
                                               //caída de tensión
          nBOR=1;
          printf("\n Reset por caida de tension!\n");      //Escritura a través
                                                           //del puerto serie
       }
```

```
        if(!nTO){
           printf("\n Reset por desbordamiento del WDT!\n");    //Escritura puerto
                                                                //serie
        }
        while(1);
}
```

Simulación

Para la simulación del programa se utilizará MPLABSIM® y se habilitará la pestaña SIM Uart de la ventana *OUTPUT* para visualizar los mensajes. En la figura inferior se muestra el valor de los registros antes de comenzar la ejecución del programa.

Update	Address	Symbol Name	Value	Decimal	Binary	Char
	00A	PCLATH	0x00	0	00000000	'.'
	002	PCL	0x00	0	00000000	'.'
	08E	PCON	0x01	1	00000001	'.'
	003	STATUS	0x1E	30	00011110	'.'

FIGURA 7.3: SIMULACIÓN DEL SISTEMA PARA LA DETECCIÓN DEL ORIGEN DE RESET PROPUESTO EN EL EJERCICIO 7.2.1 (WATCH).

Para poder observar el reset producido por el desbordamiento del *WatchDog* es necesario modificar la configuración del simulador desde el menú *Degugger/Settings*. En la ventana que aparece se deberá seleccionar la pestaña *Break Options* y en *WDT Timeout* elegir la opción *Reset* como se muestra en la figura inferior.

FIGURA 7.4: CONFIGURACIÓN DE MPLAB SIM PARA LA SIMULACIÓN DEL EJERCICIO 7.2.1.

Una vez ajustado el parámetro anterior se pueden generarán los diferentes reset desde el menú *Debugger/Reset* y ejecutar el programa mientras se observa la ventana *OUTPUT* que deberá mostrar los mensajes que aparecen en la figura inferior.

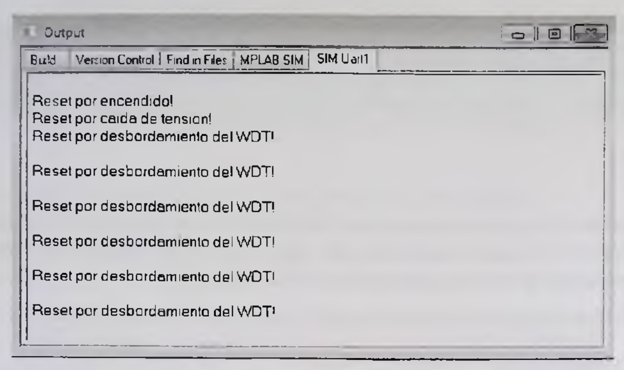


FIGURA 7.5: SIMULACIÓN DEL PROGRAMA DE DETECCIÓN DE ORIGEN DE RESET PROPUESTO EN EL EJERCICIO 7.2.1.

En caso de ejecutar indefinidamente el programa se mostrará continuamente el mensaje de desbordamiento por WDT al ser el único motivo de reset automático.

A1. Librerías en C

A1.1. ports

ports.h

```
//CONSTANTES, MACROS Y FUNCIONES PARA LOS PUERTOS DE ENTRADA Y SALIDA DIGITALES

#define  set_all_digital()      ADCON1=0x06
#define  rol_8(value,b)         value=(value<<b)|(value>>(8-b))
#define ror_8(value,b)          value=(value>>b)|(value<<(8-b))
#define bit_set(value,b) value=value|(1<<b)
#define bit_clear(value,b)      value= value&(~(1<<b))
```

A1.2. interrupts

interrupts.h

```
//CONSTANTES, MACROS Y FUNCIONES PARA LOS PUERTOS DE ENTRADA Y SALIDA DIGITALES

#define GLOBAL                  0x000080
#define PERIPHERAL              0x000040
#define INT_T0                  0x000020
#define INT_EXT                 0x000010
#define INT_PORTB               0x000080
#define INT_T1                  0x000100
#define INT_T2                  0x000200
#define INT_CCP1                0x000400
#define INT_SSP                 0x000800
#define INT_TX                  0x001000
#define INT_RX                  0x002000
#define INT_AD                  0x004000
#define INT_PSP                 0x008000
#define INT_CCP2                0x010000
#define  enable_int(value)      INTCON|=value;       \
                                PIE1|=((value&0x00FF00)>>8);     \
                                PIE2|=((value&0xFF0000)>>16)
#define  disable_int(value)     INTCON&=(~value); \
                                PIE1&=(~((value&0x00FF00)>>8)); \
                                PIE2&=(~((value&0xFF0000)>>16))
                                PIE2&=(~(value>>16))
```

A1.3. timers

timers.h

```
//CONSTANTES, MACROS Y FUNCIONES PARA EL TIMER 0

#define RTCC_INTERNAL           0x00
#define RTCC_EXT_RISE           0x20
#define RTCC_EXT_FALL           0x30
#define RTCC_DIV1               0x0F
#define RTCC_DIV2               0x00
#define RTCC_DIV4               0x01
#define RTCC_DIV8               0x02
#define RTCC_DIV16              0x03
#define RTCC_DIV32              0x04
#define RTCC_DIV64              0x05
#define RTCC_DIV128             0x06
#define RTCC_DIV256             0x07

#define setup_timer0(value) OPTION_REG=((OPTION_REG&0xC0)|value)
#define set_timer0(value)       TMR0=value
#define get_timer0()            TMR0

// CONSTANTES, MACROS Y FUNCIONES PARA EL WATCHDOG

#define WDT_18MS                0x08
#define WDT_36MS                0x09
#define WDT_72MS                0x0A
#define WDT_144MS               0x0B
#define WDT_288MS               0x0C
#define WDT_576MS               0x0D
#define WDT_1152MS              0x0E
#define WDT_2304MS              0x0F

#define setup_wdt(value) OPTION_REG=(OPTION_REG&0xF0)|value
#define restart_wdt()           CLRWDT()

//CONSTANTES, MACROS Y FUNCIONES PARA EL TIMER 1

#define T1_DISABLED             0x00
#define T1_OFF                  0x00
#define T1_ON                   0x01
#define T1_INT                  0x00
#define T1_EXT                  0x02
#define T1_DIV1                 0x00
#define T1_DIV2                 0x10
#define T1_DIV4                 0x20
#define T1_DIV8                 0x30
#define T1_OSC_EN               0x08

#define setup_timer1(value)     T1CON=value
#define set_timer1(value)       TMR1=value
#define get_timer1()            TMR1
#define start_timer1()          T1CON|=0x01
#define stop_timer1()           T1CON&=0xFE

//CONSTANTES, MACROS Y FUNCIONES PARA EL TIMER 2

#define T2_DISABLED             0x00
#define T2_OFF                  0x00
#define T2_ON                   0x04
#define T2_POST_DIV1            0x00
#define T2_POST_DIV2            0x08
#define T2_POST_DIV3            0x10
#define T2_POST_DIV4            0x18
```

```
#define T2_POST_DIV5            0x20
#define T2_POST_DIV6            0x28
#define T2_POST_DIV7            0x30
#define T2_POST_DIV8            0x38
#define T2_POST_DIV9            0x40
#define T2_POST_DIV10           0x48
#define T2_POST_DIV11           0x50
#define T2_POST_DIV12           0x58
#define T2_POST_DIV13           0x60
#define T2_POST_DIV14           0x68
#define T2_POST_DIV15           0x70
#define T2_POST_DIV16           0x78
#define T2_PRED_DIV1            0x00
#define T2_PRED_DIV4            0x01
#define T2_PRED_DIV16           0x02
#define T2_PRED_DIV64           0x03

#define setup_timer2(value)     T2CON=value
#define set_timer2(value)       PR2=value; \
                                TMR2=0
#define get_timer2()            TMR2
#define start_timer2()          T2CON|=0x04
#define stop_timer2()           T2CON&=0xFB
```

A1.4. ccp

ccp.h

```
//CONSTANTES, MACROS Y FUNCIONES PARA LOS MODULOS CCP

#define CCP_OFF                 0x00
#define   CCP_CAPTURE_FE        0x04
#define   CCP_CAPTURE_RE        0x05
#define   CCP_CAPTURE_DIV4      0x06
#define   CCP_CAPTURE_DIV16     0x07
#define   CCP_COMPARE_RISE      0x08
#define   CCP_COMPARE_FALL      0x09
#define   CCP_COMPARE_INT       0x0A
#define   CCP_COMPARE_SPECIAL   0x0B
#define   CCP_PWM               0x0C

#define   setup_ccp1(value)     CCP1CON=value
#define set_pwm1_duty(value)    CCPR1L=(value>>2);  \
                                CCP1CON=(CCP1CON&0xCF)|((value&0x0003)<<4)
#define   get_ccp1(value)       CCPR1
#define set_ccp1(value)         CCPR1=value
#define   setup_ccp2(value)     CCP2CON=value
#define set_pwm2_duty(value)    CCPR2L=(value>>2);  \
                                CCP2CON=(CCP2CON&0xCF)|((value&0x0003)<<4)
#define   get_ccp2(value)       CCPR2
#define set_ccp2(value)         CCPR2=value
```

A1.5. analog

analog.h

```
//CONSTANTES, MACROS Y FUNCIONES PARA EL CONVERSOR ANALÓGICO DIGITAL

#define ADC_OFF                 0x00
#define ALL_DIGITAL             0x06
#define ALL_ANALOG              0x00
#define ALL_ANALOG_V            0x01
#define ALL_ANALOG_V_V-         0x08
```

```
#define AN0                     0x0E
#define AN0_V_V-                0x0F
#define AN0_AN1_V               0x05
#define AN0_AN1_V_V-            0x0D
#define AN0_AN1_AN3             0x04
#define AN0_AN4                 0x02
#define AN0_AN4_V               0x03
#define AN0_AN4_V_V-            0x0C
#define AN0_AN5                 0x09
#define AN0_AN5_V               0x0A
#define AN0_AN5_V_V-            0x0B
#define _XTAL_FREQ 4000000
#define ADC_CLK_RC              0xC1
#define ADC_CLK_OSC             0X01
#define ADC_FREQ_DIV            (float)(_XTAL_FREQ/1250000)
#define START_ADC_CONV          GO=1

extern void set_vref(unsigned char vref);
extern void set_adc_inputs(unsigned char analog);
extern void setup_adc(unsigned char mode);
extern void set_adc_channel(unsigned char channel);
extern unsigned int read_adc(void);

//CONSTANTES, MACROS Y FUNCIONES PARA EL COMPARADOR ANALÓGICO

#define CA_RESET                    0x00
#define CA_A0_A3_NC_NC_ON_A4        0x01
#define CA_A0_A3_A1_A2              0x02
#define CA_A0_A3_A1_A2_ON_A4_A5     0x03
#define CA_A0_A3_A1_A3              0x04
#define CA_A0_A3_A1_A3_ON_A4_A5     0x05
#define CA_A0_VR_A1_VR              0x06
#define CA_A3_VR_A2_VR              0x0E
#define CA_NC_NC_NC_NC_OFF          0x07
#define CA_C1INV                    0x10
#define CA_C2INV                    0x20

#define SETUP_COMPARATOR(CONFIG_COMPARATOR) \
        CMCON=CONFIG_COMPARATOR;

//CONSTANTES, MACROS Y FUNCIONES PARA EL MÓDULO CVREF

#define CVRSRC              4.416
#define CVR_ON              0x80
#define  CVR_OFF            0x00
#define CVR_LOW             0x20
#define CVR_HIGH            0x00
#define CVR_OUT_ON          0x40
#define CVR_OUT_OFF         0x00
#define CVRR_0              0x00
#define CVRR_1              0x01
#define CVRR_2              0x02
#define CVRR_3              0x03
#define CVRR_4              0x04
#define CVRR_5              0x05
#define CVRR_6              0x06
#define CVRR_7              0x07
#define CVRR_8              0x08
#define CVRR_9              0x09
#define CVRR_10             0x0A
#define CVRR_11             0x0B
#define CVRR_12             0x0C
#define CVRR_13             0x0D
#define CVRR_14             0x0E
#define CVRR_15             0x0F

#define SET_CVREF(MODE)   CVRCON=MODE
extern void setup_cvref(unsigned char mode, float volt);
```

analog.c

```
#include <htc.h>
#include "analog.h"


extern void set_adc_inputs(unsigned char analog){   //Selecciona las entradas analógicas
          ADCON1=(ADCON1&0xF0)|analog;
}
extern void setup_adc(unsigned char mode){          //Selecciona el modo de reloj
       if (mode==0x00){                             //CAD apagado
          ADCON0=0x00;
          ADCON1&=0x06;    // E/S digitales
       }
       else if (mode==0xC1){                        //Reloj RC
          ADCON0=0xC1;                              //con ajuste derecho
          ADCON1&=0xBF;
       }
       else{
          if(ADC_FREQ_DIV<=1){         //Fosc <= 1.25MHz
                    ADCON0=0x01;       //ADCS1=0, ADCS0=0, Canal 0, Módulo encendido
                    ADCON1&=0xBF;      //ADFM=1(ajuste a la derecha), ADCS2=0
          }
          else if(ADC_FREQ_DIV<=2){    //Fosc <= 2.5MHz
                    ADCON0=0x01;       //ADCS1=0, ADCS0=0, Canal 0, Módulo encendido
                    ADCON1|=0xC0;      //ADFM=1(ajuste a la derecha), ADCS2=1
          }
          else if(ADC_FREQ_DIV<=4){    //Fosc <= 5MHz
                    ADCON0=0x41;       //ADCS1=0, ADCS0=1, Canal 0, Módulo encendido
                    ADCON1&=0xBF;      //ADFM=1(ajuste a la derecha), ADCS2=0
          }
          else if(ADC_FREQ_DIV<=8){    //Fosc <= 10MHz
                    ADCON0=0x41; ;     //ADCS1=0, ADCS0=1, Canal 0, Módulo encendido
                    ADCON1|=0xC0;      //ADFM=1(ajuste a la derecha), ADCS2=1
          }
          else if(ADC_FREQ_DIV<=16){   //Fosc <= 20MHz
                    ADCON0=0x81;       //ADCS1=1, ADCS0=0, Canal 0, Módulo encendido
                    ADCON1&=0xBF;      //ADFM=1(ajuste a la derecha), ADCS2=0
          }
          else{                        //Fosc <= 40MHz
                    ADCON0=0x81;       //ADCS1=1, ADCS0=0, Canal 0, Módulo encendido
                    ADCON1|=0xC0;      //ADFM=1(ajuste a la derecha), ADCS2=1
          }
       }
}

extern void set_adc_channel(unsigned char channel){ //Selecciona el canal de conversión
//Cada vez que se cambia de canal es necesario esperar el tiempo de adquisición
//(aprox 44us) para que se estabilice la tensión a la entrada del CAD antes de lanzar
//una conversión nueva.
       ADCON0 = (ADCON0&0xC7)| ((channel&0x07)<<3);
}

extern unsigned int read_adc(){               //Recoge el valor de la conversión
       while(GO)continue;
       return ((ADRESH<<2)+(ADRESL>>6));
}

extern void setup_cvref(unsigned char mode, float volt){
       if(mode&0x20) CVRCON=((int)((volt/CVRSRC)*24)&0x0F)|mode;

       else        CVRCON=((int)(((volt -(0.25*CVRSRC))/CVRSRC)*32)&0x0F)|mode;
}
```

A1.6. usart.h / usart.c

usart.h

```
#ifndef _SERIAL_H_
#define _SERIAL_H_
#define HIGH_SPEED         0x04            //Modo alta velocidad
#define LOW_SPEED          0               //Modo baja velocidad
#define NINE_BITS          0x40            //Modo transmisión 9 bits
#define EIGHT_BITS         0               //Modo transmisión 8 bits
#define _XTAL_FREQ         4000000         //Frecuencia del oscilador principal

#define RX_PIN TRISC7            //Define el pin RC7 como pin para recepción de datos
#define TX_PIN TRISC6            //Define el pin RC6 como pin para transmisión de datos

/*MACRO PARA CONFIGURAR EL MÓDULO USART EN MODO ASÍNCRONO*/
//BAUD: Velocidad de comunicaciones (9600, 19200, 28800, etc.)
//SPEED: Modo alta velocidad para el genereador de baudios => TRUE = On
//NINE: Utiliza transmission de 9 bits => FALSE=8bit
#define INIT_USART(BAUD,SPEED,NINE)\
        RX_PIN = 1; \                      //Define los pines RC6 y RC7 para poder
        TX_PIN = 1;    \                   //ser utilizados por el módulo USART
        if (SPEED)SPBRG = ((int)(_XTAL_FREQ/((16UL * BAUD) +16))); \     //Ajusta la
                                                                          //velocidad del
        else SPBRG = ((int)(_XTAL_FREQ/((64UL * BAUD) +64))); \          //gen. de baudios
        RCSTA = (NINE|0x90);     \         //Carga la configuración del módulo receptor
        TXSTA = (SPEED|NINE|0x20)          //Carga la configuración del módulo transmisor

#endif

void putch(unsigned char byte);            //Definición de la función putch
unsigned char getche(void);                //Definición de la función getche
unsigned char getch(void);                 //Definición de la función getch
```

usart.c

```
#include <htc.h>

void putch(unsigned char byte){

        while(!TXIF)                       //Se pone a '1' cuando se ha enviado el dato
           continue;
        TXREG = byte;
}

unsigned char getche() {

        while(!RCIF)                       //Se pone a '1' cuando se ha recibido un dato
           continue;
        return RCREG;
}

unsigned char getch() {

        while(!RCIF)                       //Se pone a '1' cuando se ha recibido un dato
           continue;
        return RCREG;
}
```

A1.7. i2c.h / i2c.c

i2c.h

```
#include <htc.h>

#define  WRITE    0
#define READ      1

void   i2c_init();                    //Configura el módulo i2c
void   i2c_start();                   //Envía la secuencia de start
void   i2c_restart();                 //Envía la secuencia de restart
void   i2c_stop();                    //Envía la secuencia de stop
void   i2c_idle();                    //Comprueba si el módulo i2c está libre
void   i2c_sendnack();                //Envía la secuencia de nack
unsigned char i2c_write(unsigned char byte);        //Envía un byte y devuelve 0
                                                    //en caso de error
unsigned char i2c_read();             //Lee un byte
```

i2c.c

```
#include <htc.h>
#include "i2c.h"

//Configura el modulo I2C

void   i2c_init(){
       TRISC3 = 1;                    //SLC
       TRISC4 = 1;                    //SDA
       SSPADD = 0x0A;                 //Reloj a 100KHz
       SSPSTAT = 0x80;                //Habilita el control de velocidad del módulo
       SSPCON2 = 0x00;                //Baja los flags
       SSPCON = 0x28;                 //Modo maestro, módulo I2C encendido
}

//Envía la secuencia de Start

void   i2c_start(){
       i2c_idle();                    //Comprueba que está libre
       SSPCON2bits.SEN=1;             //Envía la condición de inicio
       while(SSPCON2bits.SEN==1);     //Espera a que se envíe
       while(SSPIF==0);               //Comprueba que se ha enviado
       SSPIF=0;                       //Baja el flag
}

//Envía la secuencia de (re)start

void   i2c_restart(){
       SSPCON2bits.RSEN=1;            //Envía la secuencia restart
       while(SSPCON2bits.RSEN);       //Espera a que se envíe
       while(SSPIF==0);               //Comprueba que se ha enviado
       SSPIF=0;                       //Baja el flag
}
//Envía la secuencia de Stop

void   i2c_stop(){
       SSPCON2bits.PEN=1;             //Envía la secuencia de Stop
       while(SSPCON2bits.PEN);        //Espera a que se envíe
       while(SSPIF==0);               //Comprueba que se ha enviado
       SSPIF=0;                       //Baja el flag
}
```

```
//Comprueba que el modulo I2C está libre

Void i2c_idle(){
while( ( SSPCON2 & 0x1F ) | SSPSTATbits.R_nW );     //Comprueba que el módulo está libre
}

//Lee un byte

unsigned char i2c_read(){
        i2c_idle();
        SSPCON2bits.RCEN=1;              //Activa el modo recepción
        while(SSPCON2bits.RCEN==1);      //Espera a recibir el byte
        while(SSPSTATbits.BF==0);        //Espera a que el buffer esté lleno
        return SSPBUF;                   //Devuelve el dato
}

//Envía un byte al esclavo y devuelve 0 en caso de error

unsigned char i2c_write(unsigned char byte){
        i2c_idle();
        SSPBUF=byte;
        while(SSPSTAT&0x05);             //Comprueba si hay transmisión en curso
        if(SSPCON2bits.ACKSTAT==1){      //No ha recibido confirmación
           SSPCON2bits.PEN=1;            //Envía secuencia de Stop
           return  0 ;                   //devuelve 0
        }
        else{                            //Ha recibido confirmación
           while(SSPIF==0);              //Comprueba que ha transmitido
           SSPIF=0;                      //Baja el flag
           return  1;                    //Devuelve 1
        }
}

//Envía la secuencia Nack

void   i2c_sendnack(){
        SSPCON2bits.ACKDT=1;
        SSPCON2bits.ACKEN=1;
        while(SSPCON2bits.ACKEN==1);
}
```

A1.8. lcd.h / lcd.c

lcd.h

```
/* Escribe un byte en la pantalla LCD en modo 4 bits */
extern void lcd_write(unsigned char);
/* Limpia la pantalla LCD */
extern void lcd_clear(void);
/* Escribe una cadena de caracteres en la pantalla LCD */
extern void lcd_puts(const char * s);
/* Posiciona el cursor en la posición indicada */
extern void lcd_goto(unsigned char pos);
/* Inicializa la pantalla LCD - utilizar anter de cualquier otra función */
extern void lcd_init(void);
```

```
/* Escribe en la pantalla LCD el carácter cuyo código ASCII se indica */

extern void lcd_putch(char);

/* Establece la posición del cursor */

#define   lcd_cursor(x) lcd_write(((x)&0x7F)|0x80)
```

lcd.c

```
/*
 * Rutinas para trabajar con la pantalla LCD Hitachi HD44780.
 * Utiliza comunicaciones en modo 4 bit de la siguiente forma:
 *
 * PORTD bits 0-3 conectados a los bits de datos 4-7 de la LCD (high nibble)
 * PORTD bit 4 conectado a la entrada LCD RS (selección de registro)
 * PORTD bit 5 conectado a la entrada LCD RW (lectura/escritura)
 * PORTD bit 6 conectado a la entrada LCD EN (reloj)
 * PORTD bit 7 conectado a la entrada LCD ON bit (LCD power)
 */

#ifndef _XTAL_FREQ
//Se asume una frecuencia de trabajo de 4MHz a menos que se haya especificado
//con anterioridad
#define _XTAL_FREQ 4000000
#endif

#include <htc.h>
#include "lcd.h"

#define LCD_RS RD4
#define LCD_RW RD5
#define LCD_EN RD6

#define LCD_DATA PORTD

#define LCD_STROBE() ((LCD_EN = 1),(LCD_EN=0))

/* envía un byte a la LCD en modo 4 bits */

void lcd_write(unsigned char c){
        __delay_us(40);
        LCD_DATA = (LCD_DATA & 0xF0) | ( ( c >> 4 ) & 0x0F );
        LCD_STROBE();
        LCD_DATA = (LCD_DATA & 0xF0) | ( c & 0x0F );
        LCD_STROBE();
}

/* Limpia la pantalla LCD */
void lcd_clear(void){
        LCD_RS = 0;
        lcd_write(0x1);
        __delay_ms(2);
}

/* envía una cadena de caracteres a la LCD */
void lcd_puts(const char * s){
        LCD_RS = 1; // write characters
        while(*s)
           lcd_write(*s++);
}

/* escribe un character en la LCD */
void lcd_putch(char c){
        LCD_RS = 1; // write characters
        lcd_write( c );
}
```

```
/* Desplaza el cursor a la posición indicada */
void lcd_goto(unsigned char pos){
       LCD_RS = 0;
       lcd_write(0x80+pos);
}

/* Inicializa la pantalla LCD en modo 4 bits */
void lcd_init(){
       char init_value;

       init_value = 0x3;
       TRISD=0;
       RD7=1;                                          //enciende la pantalla LCD
       LCD_RS = 0;
       LCD_EN = 0;
       LCD_RW = 0;
       __delay_ms(15);                                 //espera 15ms
       LCD_DATA = (LCD_DATA & 0xF0) | init_value;      //envía el commando de inicio
       LCD_STROBE();
       __delay_ms(5);
       LCD_STROBE();
       __delay_us(200);
       LCD_STROBE();
       __delay_us(200);
       LCD_DATA = (LCD_DATA & 0xF0) | 2;               //Modo 4 bits
       LCD_STROBE();

       lcd_write(0x28);                     //Longitud del caracter
       lcd_write(0xF);                      //Display On, Cursor On, Cursor Blink
       lcd_clear();                         //Limpia la pantalla
       lcd_write(0x6);                      //Modo de entrada On
}
```

A2. Registros de Funciones Especiales (SFRs)

A2.1. STATUS

Registro STATUS (direcciones 03h, 83h, 103h, 183h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	TO#	PD#	Z	DC	C
bit 7							bit 0

Valores de los bits después de un reset
R = el bit se puede leer
W = el bit se puede escribir
x = Valor desconocido

Nota: TO# = $\overline{TO}$

bit 7 IRP**:** Selecciona el Banco de Memoria de datos en el direccionamiento indirecto.
0 = Bancos 0 y 1 (00h - FFh) (256 bytes)
1 = Bancos 2 y 3 (100h - 1FFh) (256 bytes)

bit 6-5 RP1:RP0**:** Seleccionan el Banco de Memoria de datos en el direccionamiento directo.
00 = Banco 0 (00h - 7Fh) (128 bytes)
01 = Banco 1 (80h - FFh) (128 bytes)
10 = Banco 2 (100h - 17Fh) (128 bytes)
11 = Banco 3 (180h - 1FFh) (128 bytes)

bit 4 TO# ***(Time-out)*:** Indicador de desbordamiento del perro guardián WTD *(Watchdog Timer)*
0 = Cuando se desborda el WTD
1 = Después de un reset por encendido (*power-up*) y con las instrucciones CLRWDT y SLEEP

bit 3 PD# **(*Power-down*):** Indicador de modo de bajo consumo.
0 = Cuando entra en bajo consumo con la instrucción SLEEP
1 = Después del encendido o con la instrucción CLRWDT

bit 2 Z ***(Zero)*:** Indicador de cero
1 = Si el resultado de una operación aritmética o lógica es cero
0 = Si el resultado de una operación aritmética o lógica es no es cero

bit 1 DC ***(Digit carry/borrow)*:** Indicador de acarreo o préstamo auxiliar en las operaciones aritméticas de suma o resta (instrucciones ADDWF, ADDLW, SUBLW, SUBWF)
1 = Si hay acarreo del bit 3 al 4 en el resultado de una operación aritmética de suma binaria o si no hay préstamo en una operación de resta
0 = Si no hay acarreo en la suma o si hay préstamo del bit 4 al bit 3 en una operación de resta

bit 0 C ***(Carry/borrow)*:** Indicador de acarreo o préstamo en las operaciones aritméticas de suma o resta (instrucciones ADDWF, ADDLW, SUBLW, SUBWF)
1 = Si hay acarreo en el resultado de una operación aritmética de suma binaria o si no hay préstamo en una operación de resta
0 = Si no hay acarreo en la suma o si hay prestamo en una operación de resta

A2.2. Registro OPTION_REG

Registro OPTION_REG (direcciones 0x81h, 0x181h de la memoria de datos RAM)

R/W-1	RW-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU#	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

bit 7 RBPU#: Habilita/deshabilita las resistencias internas de Pull-up del puerto B.

1 = PORTB pull-ups deshabilitadas
0 = PORTB pull-ups habilitadas

bit 6 INTEDG: Selección del planco para la generación de la interrupción externa por RB0.

1 = Interrupción externa RB0/INT por flanco de subida
0 = Interrupción externa RB0/INT por flanco de bajada

bit 5 T0CS: configura al Timer0 como temporizador (T0CS = 0) o como contador (T0CS = 1).
bit 4 T0SE: configura el flanco de la señal externa con el que se incrementa el Timer0 si ha sido programado como contador (T0SE = 0 para flanco de subida y TS0E = 1 para flanco de bajada).

bit 3 PSA: asigna el pre-divisor al Timer0 (PSA = 0) o al Perro Guardián WDT (PSA =1)

Bits 2, 1 y 0 PS2:PS1:PS0: programan el factor de división del pre-divisor.

PS2 PS1 PS0	Factor división para TMR0	Factor división para WDT
000	2	1
001	4	2
010	8	4
011	16	8
100	32	16
101	64	32
110	128	64
111	256	128

A2.3. T1CON

Registro T1CON (dirección 0x10 de la memoria de datos RAM)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
-	-	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC#	TMR1CS	TMR1ON
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

bit 7-6 No implementados: se leerían como '0'.

bit 5-4 T1CKPS1:T1CKPS0: Bits de selección del factor de división del pre-divisor para la entrada de reloj del Timer1

T1CKPS1-T1CKPS0	Factor de división
00	1
01	2
10	4
11	8

bit 3 T1OSCEN: Bit de habilitación (enable) del oscilador externo (T1OSC) del TMR1
1 = Oscilador habilitado
0 = Oscilador apagado

bit 2 T1SYNC: Bit de control de la sincronización del reloj externo del Timer1
Cuando TMR1CS=1:
1 = No sincroniza la entrada de reloj externo
0 = Sincroniza la entrada de reloj externo
Cuando TMR1CS = 0:
Este bit se ignora, pues TIMER1 usa el reloj interno que ya está sincronizado.

bit 1 TMR1CS: Bit de selección de reloj para el TIMER 1
1 = Reloj externo del pin RC0/T1OSO/T1CKI (flancos de subida)
0 = Reloj interno (FOSC/4)

bit 0 TMR1ON: Bit para arranque/paro del Timer1.
1 = Timer1 cuenta pulsos
0 = Timer1 parado

A2.4. T2CON

Registro T2CON (dirección 0x12 de la memoria de datos RAM)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

bit 7 No implementado: **Se lee como 0**

bit 6:3 TOUTPS3:TOUTPS0: **Bits de selección del factor de división del post-divisor del Timer2**

TOUTPS3-TOUTPS0	Factor de división
0000	1
0001	2
0010	3
0011	4
0100	5
0101	6
0110	7
0111	8
1000	9
1001	10
1010	11
1011	12
1100	13
1101	14
1110	15
1111	16

bit 2 TMR2ON: **Bit de paro/arranque del TMR2**
1 = Timer2 on (habilitado)
0 = Timer2 off (no habilitado)

bit 1:0 T2CKPS1:T2CKPS0: **Bits de selección del factor de división del pre-divisor del Timer2**

T2CKPS1-T2CKPS0	Factor de división
00	1
01	4
1x	16

A2.5. Registro INTCON

Registro INTCON (direcciones 0Bh, 8Bh, 10Bh, 18Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
bit 7							bit 0

bit 7 GIE *(Global Interrupt Enable):* Habilitación del sistema de interrupciones del microcontrolador
1 = Se habilitan todas las interrupciones no enmascaradas
0 = Se inhabilitan todas las interrupciones.

bit 6 PEIE *(Peripheral Interrupt Enable):* Habilitación de las interrupciones de los periféricos (en registros PIE1,PIR,PIE2, PIR2)
1 = Se habilitan todas las interrupciones de los periféricos no enmascaradas
0 = Se inhabilitan todas las interrupciones de periféricos.

bit 5 TOIE *(TMR0 Overflow Interrupt Enable):* Habilitación de la interrupción por desbordamiento del Timer0
1 = Se habilita la interrupción
0 = Se inhabilita la interrupción

bit 4 INTE *(RB0/INT External Interrupt Enable)*: Habilitación de la interrupción externa
1 = Habilita la interrupción externa por RB0/INT
0 = Inhabilita la interrupción externa por RB0/INT

bit 3 RBIE *(RB Port Change Interrupt Enable)*: Habilitación de la interrupción por cambio en cualquier terminal RB4-RB7
1 = Se habilita la interrupción
0 = Se inhabilita la interrupción

bit 2 TOIF *(TMR0 Overflow Interrupt Flag)*: Señalizador del desbordamiento del timer0
1 = El registro TMR0 se ha desbordado (se debe limpiar, desactivar por software)
0 = el registro TMR0 no se ha desbordado

bit 1 INTF *(RB0/INT External Interrupt Flag)*: Señalizador de petición de interrupción externa
1 = Se produce petición de interrupción (se debe desactivar por software)
0 = No se produce petición de interrupción externa

bit 0 RBIF (RB Port Change Interrupt Flag): Señalizador de cambio en algún terminal RB4-RB7
1 = Al menos un terminal RB7-RB4 ha cambiado de estado (se debe desactivar por software).
0 = Ningún terminal RB7-RB4 ha cambiado de estado

A2.6. Registro PIE1

Registro PIE1
(dirección 8Ch)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE[1]	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

bit 7 PSPIE[1] : Permiso de interrupción para el puerto paralelo Esclavo (PSP) al realizar una operación de lectura /escritura 1 = Habilita la interrupción 0 = Inhabilita la interrupción **bit 6** ADIE: Permiso de interrupción para el conversor A/D al finalizar la conversión 1 = Habilita la interrupción 0 = Inhabilita la interrupción **bit 5** RCIE: Permiso de interrupción para el receptor del USART cuando el buffer se llena 1 = Habilita la interrupción 0 = Inhabilita la interrupción **bit 4** TXIE: Permiso de interrupción para el transmisor del USART cuando el buffer se vacía 1 = Habilita la interrupción 0 = Inhabilita la interrupción	**bit 3** SSPIE: Permiso de interrupción para el puerto serie síncrono (SSP) 1 = Habilita la interrupción 0 = Inhabilita la interrupción **bit 2** CCP1IE: Permiso de interrupción para el módulo CCP1 cuando se produce una captura o comparación 1 = Habilita la interrupción 0 = Inhabilita la interrupción **bit 1** TMR2IE: Permiso de interrupción para el TMR2 con su desbordamiento. 1 = Habilita la interrupción 0 = Inhabilita la interrupción **bit 0** TMR1IE: Permiso de interrupción para el TMR1 con su desbordamiento. 1 = Habilita la interrupción 0 = Inhabilita la interrupción **Nota: PSPIE no existe en los PIC16F873/873A/876/876A (modelos de 28 terminales que no tienen el PSP)**

A2.7. Registro PIR1

Registro PIR1
(dirección 0Ch)

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF[1]	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

bit 7 PSPIF[1] : Señalizador (de interrupción) del puerto paralelo Esclavo (PSP) al realizar una operación de lectura /escritura 1 = Se ha producido una operación de R/W (desactivar por software) 0 = No se ha producido operación de R/W. **bit 6** ADIF: Señalizador (de interrupción) del conversor A/D al finalizar la conversión 1 = La conversión A/D está completada 0 = La conversión A/D no está completada **bit 5** RCIF: Señalizador (de interrupción) del receptor del USART cuando el buffer se llena 1 = Buffer lleno 0 = Buffer no lleno **bit 4** TXIF: Señalizador (de interrupción) del transmisor del USART cuando el buffer se vacía 1 = Buffer vacío 0 = Buffer no vacío **bit 3** SSPIF: Señalizador (de interrupción) del puerto serie síncrono (SSP) 1 = Se ha producido la condición de interrupción del SSP, es decir, se ha producido una transmisión/recepción (desactivar por software antes de salir de la RAI) 0 = No se produce condición de in6terrupción de SSP	**bit 2** CCP1IF: Señalizador (de interrupción) del módulo CCP1 MODO CAPTURA: 1 = El registro del CCP captura el valor del registro TMR1 (valor del Timer1) (desactivar por software) 0 = No se produce la captura del registro TMR1 MODO COMPARACIÓN: 1 = Coinciden el valor del CCP con el valor del TMR1 0 = No coinciden MODO PWM (modulación de pulsos en anchura): no se usa **bit 1** TMR2IF: Señalizador (de interrupción) por coincidencia de TMR2 con PR2 1 = Se produce la coincidencia 0 = No se produce la coincidencia **bit 0** TMR1IF: Señalizador (de interrupción) por desbordamiento (overflow) del TMR1 1 = Desbordamiento del TMR1 (desactivar por software) 0 = No desbordamiento del registro TMR1 Note 1: PSPIE no existe en los PIC16F873/873A/876/876A (modelos de 28 terminales que no tienen el PSP)

A2.8. Registro PIE2 y PIR2

Registro PIE2 (dirección 8Dh)

U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
-	CMIE	-	EEIE	BCLIE	-	-	CCP2IE
bit 7							bit 0

bit 7,5,2,1 Unimplemented: No implementado. Se lee '0'
bit 6 CMIE: Permiso de interrupción para el Comparador
1 = Habilita la interrupción
0 = Inhabilita la interrupción
bit 4 EEIE: Permiso de interrupción por fin de escritura en la EEPROM de datos
1 = Habilita la interrupción
0 = Inhabilita la interrupción
bit 3 BCLIE: Permiso de interrupción por colisión de Bus en el el puerto serie síncrono (SSP) cuando dos o más maestros tratan de transferir al mismo tiempo.
1 = Habilita la interrupción
0 = Inhabilita la interrupción
bit 0 CCP2IE: Permiso de interrupción en el módulo CCP2
1 = Habilita la interrupción
0 = Inhabilita la interrupción

Registro PIR2 (dirección 0Dh)

U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
-	CMIF	-	EEIF	BCLIF	-	-	CCP2IF
bit 7							bit 0

bit 7,5,2,1 Unimplemented: No implementado. Se lee '0'
bit 6 CMIF: Flag(de interrupción) del comparador
1 = Ha cambiado la entrada del comparador
0 = No ha cambiado
bit 4 EEIF: Flag (de interrupción) del fin de escritura en la EEPROM
1 = Se ha completado la operación de escritura (desactivar por software)
0 = No se ha completado o iniciado la operación de escritura
bit 3 BCLIF: Flag (de interrupción) por colisión de BUS.
1 = Ha ocurrido una colisión de bus en el SSP.
0 = No ha ocurrido la colisión
bit 0 CCP2IF: Flag (de interrupción) del módulo CCP2
MODO CAPTURA:
1 = El registro del CCP captura el valor del registro TMR1 (valor del Timer1) (desactivar por software)
0 = No se produce la captura del registro TMR1
MODO COMPARACIÓN:
1 = Coinciden el valor del CCP con el valor del TMR1
0 = No coinciden
MODO PWM (modulación de pulsos en anchura): no se usa

A2.9. CCPxCON

Bits 3-0	CCPxM3: CCPxM0		Bits de selección del modo de funcionamiento del módulo CCPx	
	00	00	Módulo CCPx desactivado (resetea el módulo CCPx)	
	01	00	Modo Captura	Captura cada flanco de bajada
		01	Modo Captura	Captura cada flanco de subida
		10	Modo Captura	Captura cada 4 flancos de subida
		11	Modo Captura	Captura cada 16 flancos de subida
	10	00	Modo Comparación	El terminal CCPx es iniciado en bajo '0' y se pone en alto '1' cuando el resultado de la comparación es positivo, es decir, cuando TMR1 alcanza el valor del registro de16 bits (CCPRxH.CCPRxL). El bit CCPxIF se activa y se pone a 1
		01	Modo Comparación	El terminal CCPx es iniciado en alto '1' y se pone en bajo '0' cuando el resultado de la comparación es positivo, es decir, cuando TMR1 alcanza el valor del registro de16 bits (CCPRxH.CCPRxL). El bit CCPxIF se activa y se pone a 1
		10	Modo Comparación	El bit CCPIF es puesto a 1 cuando el resultado de la comparación es positivo, es decir, generación de interrupción software cuando se produce la igualdad. El terminal CCPx no se ve afectado
		11	Modo Comparación	Generación de disparo de evento especial (special event trigger) cuando el resultado de la comparación es positivo. El bit CCPxIF se pone a 1. El terminal CCPx no se ve afectado. CCP1 resetea el TMR1; CCP2 resetea el TMR1 y lanza una conversión A/D nueva (si el módulo del conversor A/D está habilitado.
	11	xx	PWM	
Bits 5-4	DCxB1:DCxB0 (CCPxX:CCPxY)		Modo PWM	Bits 0 y 1 del valor que fija el ciclo de trabajo en el modo PWM En los modos CAPTURA y COMPARACIÓN NO SE UTILIZAN
Bits 7-6	No implementados			Se leerían como '0'

A2.10. ADCON0

REGISTRO ADCON0 (dirección 1Fh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE#	-	ADON
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

bit 7-6 ADCS1:ADCS0: Selección del reloj para la conversión A/D (en los PIC16F87x y versiones "antiguas") y junto con ADCS2 que está en ADCON1 (en los PIC16F87x A)

ADCS1:ADCS0	Fuente de reloj	Frecuencia del reloj del convertidor A/D	
		ADCS2 = 0	ADCS2 = 1
00	Oscilador principal	Fosc/2	Fosc/4
01	Oscilador principal	Fosc/8	Fosc/16
10	Oscilador principal	Fosc/32	Fosc/64
11	Oscilador RC interno	167 KHz a 500 KHz	167 KHz a 500 KHz

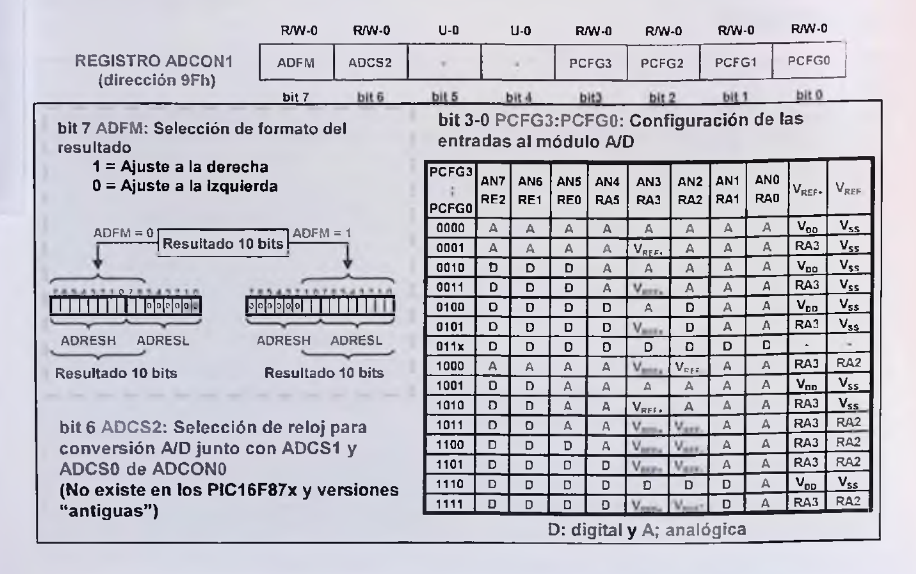
bit 5-3 CHS2:CHS0:
Selección del canal de conversión

000 = Canal 0   100 = Canal 4
001 = Canal 1   101 = Canal 5
010 = Canal 2   110 = Canal 6
011 = Canal 3   111 = Canal 7

bit 2 GO/DONE#:
Estado de la conversión
Si ADON=1:
1 = Conversión en progreso
0 = Conversión finalizada

bit 0 ADON:
Bit de encendido del convertidor A/D
1 = Módulo A/D encendido
0 = Módulo A/D apagado

A2.11. ADCON1



REGISTRO ADCON1 (dirección 9Fh)

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	-	-	PCFG3	PCFG2	PCFG1	PCFG0
bit 7	bit 6	bit 5	bit 4	bit3	bit 2	bit 1	bit 0

bit 7 ADFM: Selección de formato del resultado
1 = Ajuste a la derecha
0 = Ajuste a la izquierda

bit 6 ADCS2: Selección de reloj para conversión A/D junto con ADCS1 y ADCS0 de ADCON0
(No existe en los PIC16F87x y versiones "antiguas")

bit 3-0 PCFG3:PCFG0: Configuración de las entradas al módulo A/D

PCFG3 : PCFG0	AN7 RE2	AN6 RE1	AN5 RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	V_{REF+}	V_{REF}
0000	A	A	A	A	A	A	A	A	V_{DD}	V_{SS}
0001	A	A	A	A	V_{REF+}	A	A	A	RA3	V_{SS}
0010	D	D	D	A	A	A	A	A	V_{DD}	V_{SS}
0011	D	D	D	A	V_{REF+}	A	A	A	RA3	V_{SS}
0100	D	D	D	D	A	D	A	A	V_{DD}	V_{SS}
0101	D	D	D	D	V_{REF+}	D	A	A	RA3	V_{SS}
011x	D	D	D	D	D	D	D	D	-	-
1000	A	A	A	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2
1001	D	D	A	A	A	A	A	A	V_{DD}	V_{SS}
1010	D	D	A	A	V_{REF+}	A	A	A	RA3	V_{SS}
1011	D	D	A	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2
1100	D	D	D	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2
1101	D	D	D	D	V_{REF+}	V_{REF-}	A	A	RA3	RA2
1110	D	D	D	D	D	D	D	A	V_{DD}	V_{SS}
1111	D	D	D	D	V_{REF+}	V_{REF-}	D	A	RA3	RA2

D: digital y A; analógica

A2.12. CVRCON

REGISTRO CVRCON (dirección 9Dh)

R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
CVREN	CVROE	CVRR	.....	CVR3	CVR2	CVR1	CVR0
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Bit 7 CVREN: Bit para la habilitación de la referencia de tensión del comparador ('1' habilitado, '0' deshabilitado).

1 = Tensión de referencia CV_{REF} habilitada
0 = Tensión de referencia CV_{REF} deshabilitada

bit 6 CVROE: Bit para la habilitación de la referencia de tensión de salida del comparador
1 = Tensión de salida CV_{REF} por RA2
0 = Tensión de salida CV_{REF} por RA2 desconectada

Bit 5 CVRR: Bit de selección del rango de salida CV_{REF} ('1' rango bajo, '0' rango alto)

1 = de 0 a 0.75 CV_{RSRC} cada $CV_{RSRC}/24$
0 = de 0.25 a 0.75 CV_{RSRC} cada $CV_{RSRC}/32$

bit 3-0 CVR3:CVR0: Bits para la selección del valor de referencia V_{REF} del comparador ($CV_{RSRC}=V_{DD}$)

Cuando CVRR = 1
$CV_{REF} = (VR<3:0>/24)\cdot(CV_{RSRC})$
Cuando CVRR = 0
$CV_{REF} = 1/4\cdot(CV_{RSRC}) + (VR<3:0>/32)\cdot(CV_{RSRC})$

A2.13. CMCON

REGISTRO CMCON (dirección 9Ch)

R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-1	R/W-1
C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Bit 7 CxOUT: Salida del comparador x

Cuando CxINV = 0	Cuando CxINV = 1
1 si Cx Vin+ > Cx Vin-	1 si Cx Vin+ < Cx Vin -
0 si Cx Vin + < Cx Vin-	0 si Cx Vin+ > Cx Vin-

bit 6 CxINV: Bit para invertir la salida del comparador x
1 = Salida Cx invertida
0 = Salida Cx no invertida

Bit 5 CIS: Selección de entrada del comparador (cuando CM2:CM0 = 110)

1 = C1 Vin- conectado a RA3/AN3
C2 Vin- conectado a RA2/AN2
0 = C1 Vin- conectado a RA0/AN0
C2 Vin- conectado a RA1/AN1

bit 3-0 CM2:CM0: Modos de funcionamiento de los comparadores.

MODO	CM2:CM0
1. C1 & C2 deshabilitados.	000
2. C1 independiente; C2 apagado.	001
3. C1 & C2 independientes.	010
4. C1 & C2 independientes con salidas RA4/RA5 habilitadas.	011
5. C1 & C2 con referencia común por RA3/RA2.	100
6. C1 & C2 con referencia común por RA3/RA2 y salidas RA4/RA5 hablitadas.	101
7. Entradas multiplexadas RA0/RA3 y RA1/RA2 con referencia de tensión común por CVREF.	110
8. C1 & C2 apagados.	111

A2.14. TXSTA

REGISTRO TXSTA (dirección 98h)

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Bit 7 CSRC: Bit para la seleccion de reloj
<u>Modo asíncrono</u>
Se ignora
<u>Modo síncrono</u>
1 = Master (reloj generado por el módulo BRG)
0 = Slave (reloj externo)
Bit 6 TX9 Habilitación de la transferencia en modo 9 bits
1 = Transmisión 9 bits
0 = Transmisión 8 bits
Bit 5 TXEN Habilitación de la transmisión
1 = Transmisión habilitada
0 = Transmisión deshabilitada
Bit 4 SYNC Selección de modo
1 = Síncrono
0 = Asíncrono
Bit 2 BRGH Selección del modo de funcionamiento del generador interno.
<u>Modo asíncrono</u>
1 = Alta velocidad
0 = Baja velocidad
<u>Modo síncrono</u>
Se ignora
Bit 1 TRMT: Indicador de estado del registro de desplazamiento de transmisión
1 = Registro TSR vacío
0 = Registro TSR lleno
Bit 0 TX9D. Noveno bit de transmisión de datos (puede ser el bit de paridad).

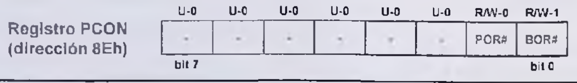
A2.15. RCSTA

REGISTRO RCSTA (dirección 18h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Bit 7 SPEN: Bit para la selección de reloj.
1 = USART ON
0 = USART OFF
Bit 6 RX9: Habilitación de la recepción en modo 9 bits
1 = Tx 9 bits 0 = Tx 8 bits
Bit 5 SREN Recepción de 1 bit
<u>Modo asíncrono</u>
Se ignora
<u>Modo síncrono - Master</u>
1 = Habilitada 0 = Deshabilitada
<u>Modo síncrono – Slave</u>
Se ignora
Bit 4 CREN Recepción continua
<u>Modo asíncrono</u>
1 = Habilitada 0 = Deshabilitada
<u>Modo síncrono</u>
1 = Habilitada mientras hasta que CREN es 0
0 = Deshabilitada
Bit 3 ADDEN Detección de dirección
<u>Modo asíncrono 9-bits (RX9=1)</u>
1 = Permite detección de dirección, habilita la interrupción y la carga del buffer de recepción cuando RSR<8> esta a '1
0 = Deshabilita la detección de dirección, se reciben todos los bits y el bit 9 puede utilizarse como bit de paridad
Bit 2 FERR Indicador de error de trama
1 = Error
0 = Sin error
Bit 1 OERR: Error por desbordamiento del regisIro de desplazamiento
1 = Error
0 = Sin error
Bit 0 RX9D Noveno bit de recepción de datos (puede ser el bit de paridad)

A2.16. PCON



bit 7-2 Unimplemented: No implementado. Se lee como '0'
bit 1 POR ***(Power-on Reset Status bit):*** Indica si el reset es por encendido o conexión de la alimentación.
0 = Cuando se produce el reset por encendido (debe ponerse a 1 por software después de que ocurra el reset por encendido)
1 = No se produce reset por encendido
bit 0 BOR ***(Brown-out Reset Status bit):*** Indica si el reset es por caida de tensión.
0 = Cuando se produce el reset por caida de tensión (debe ponerse a 1 por software después de que ocurra el reset por caida de tensión)
1 = No se produce reset por caida de tensión

A2.17. SSPSTAT

MODO SPI

Registro SSPSTAT (dirección 94h)

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE	$D/\overline{A}$	P	S	$R/\overline{W}$	UA	BF
bit 7							bit 0

Valores de los bits después de un reset
R = el bit se puede leer
W = el bit se puede escribir
x = Valor desconocido

bit 7 SMP **(Sample bit)**: Modo de muestreo de datos.

SPI Maestro:
1 = Muestreo de datos al final del tiempo de salida de datos
0 = Muestreo de datos a mitad del tiempo de salida de datos
SPI Esclavo
Se debe poner a '0' cuando funciona en modo esclavo

bit 6 CKE (***SPI clock select bit***): Seleccionan el modo de transmisión de datos.

1 = Transmite cuando la señal de reloj pasa de '1' a '0'.

0 = Transmite cuando la señal de reloj pasa de '0' a '1'.

bit 5 $D/\overline{A}$ ***(Data/Address bit)***

Se utiliza sólo en modo I^2C.

bit 4 P ***(Stop bit)***: Bit de parada

Se utiliza sólo en modo I^2C. Se pone a 0 cuando el módulo MSSP está deshabilitado (SSPEN=0)

bit 3 S ***(Start bit)***: Bit de inicio

Se utiliza sólo en modo I^2C.

bit 2 $R/\overline{W}$ ***(Read/Write information bit)***

Se utiliza sólo en modo I^2C.

bit 1 UA ***(Update address bit)***

Se utiliza sólo en modo I^2C

bit 0 BF ***(Buffer full status bit)***: Bit indicador de buffer lleno.

Sólo en modo recepción

1 = Recepción completa, SSPBUF lleno

0 = Recepción incompleta, SSPBUF vacio

MODO I^2C

Registro SSPSTAT (dirección 94h)

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE	$D/\overline{A}$	P	S	$R/\overline{W}$	UA	BF
bit 7							bit 0

Valores de los bits después de un reset
R = el bit se puede leer
W = el bit se puede escribir
x = Valor desconocido

bit 7 SMP **(Slew rate control bit)**: Control de velocidad

Maestro o esclavo
1 = Control habilitado para velocidades estándar (100KHz y 1 MHz)
0 = Control deshabilitado para modo alta velocidad (400 KHz).

bit 6 CKE ***(SMBUS Select bit)***
1 = Habilita entradas específicas SMBus.
0 = Deshabilita entradas específicas SMBus.

bit 5 $D/\overline{A}$ ***(Data/Address bit)***

Reservado en modo maestro

Esclavo
1 = El último byte recibido es un dato.
0 = El último byte recibido es una dirección.

bit 4 P ***(Stop bit)***: Bit de parada.
1 = Indica que se ha recibido un bit de stop.
0 = Indica que no se ha detectado bit de stop.
Se pone a '0' en un Reset o al apagar el módulo

bit 3 S ***(Start bit)***: Bit de inicio
1 = Indica que se ha recibido un bit de inicio.
0 = Indica que no se ha detectado bit de inicio.
Se pone a '0' en un Reset o al apagar el módulo

bit 2 $R/\overline{W}$ ***(Read/Write information bit)***

Esclavo	Maestro
1 = Lectura.	1 = Transmisión en curso
0 = Escritura	0 = No hay transmisión en curso

bit 1 UA ***(Update address bit)***: Sólo para modo 10-bit esclavo)
1 = El usuario debe actualizar la dirección SSPADD.
0 = No se necesita actualizar la dirección SSPADD

bit 0 BF ***(Buffer full status bit)***: Bit indicador de buffer lleno.
Modo Transmisión
1 = Recepción completa, SSPBUF lleno
0 = Recepción incompleta, SSPBUF vacio
Modo Recepción
1 = Transmisión en proceso, SSPBUF lleno
0 = Transmisión completa, SSPBUF vacio

A2.18. SSPCON

MODO SPI

Registro SSPCON (dirección 14h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7							bit 0

Valores de los bits después de un reset
R = el bit se puede leer
W = el bit se puede escribir
x = Valor desconocido

bit 7 WCOL *(Write Collision Detect bit)*: bit indicador de colisión.
1 = Colisión en el envío de datos (debe limpiarse por software).
0 = No hay colisión.

bit 6 SSPOV *(Receive Overflow Indicator bit)*: bit indicador de desbordamiento.
SPI esclavo
1 = Hay colisión en la recepción de datos (debe limpiarse por software.
0 = No hay desbordamiento.

bit 5 SSPEN *(Synchronous Serial Port Enable bit)*: Habilita el módulo MSSP.
1 = Habilita el módulo MSSP y asocia los terminales SCK, SDO, SDI y $\overline{SS}$ al módulo.
0 = Deshabilita el módulo MSSP y configura los terminales como puertos E/S.

Cuando se habilita el módulo sus pines se deben configurar adecuadamente como entradas/salidas

bit 4 CKP *(Clock polarity select bit)*: Bit de selección de polaridad del reloj.

1 = Estado alto '1' inactivo.
0 = Estado bajo '0' inactivo.

bits 3:0 SSPM3:SSPM0 *(Synchronous serial port mode select bits)*: Bits de selección del modo de funcionamiento del módulo SPI

0101 = SPI esclavo clock = SCK, control $\overline{SS}$ deshabilitado

0100 = SPI esclavo clock = SCK, control $\overline{SS}$ habilitado

0100 = SPI maestro, clock = TMR2/2

0100 = SPI maestro, clock = Fosc / 64

0100 = SPI maestro, clock = Fosc / 16

0100 = SPI maestro, clock = Fosc / 4

El resto de combinaciones están reservadas o se utilizan únicamente en modo I²C

MODO I²C

Registro SSPCON (dirección 14h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7							bit 0

Valores de los bits después de un reset
R = el bit se puede leer
W = el bit se puede escribir
x = Valor desconocido

bit 7 WCOL *(Write Collision Detect bit)*: bit indicador de colisión.
Transmisión Master/Slave
1 = Colisión en el envío de datos (debe limpiarse por software).
0 = No hay colisión.

bit 6 SSPOV *(Receive Overflow Indicator bit)*: bit indicador de desbordamiento.
Modo recepción
1 = Hay colisión en la recepción de datos (debe limpiarse por software.
0 = No hay desbordamiento.

bit 5 SSPEN *(Synchronous Serial Port Enable bit)*: Habilita el módulo MSSP.
1 = Habilita el módulo MSSP y asocia los terminales SDA, SCL al módulo.
0 = Deshabilita el módulo MSSP y configura los terminales como puertos E/S.

Cuando se habilita el módulo sus pines SDA y SCL se deben configurar adecuadamente como entradas/salidas

bit 4 CKP *(SCK release control bit)*: Bit de liberación de reloj.

1 = Libera reloj.
0 = Mantiene el reloj en bajo.

bits 3:0 SSPM3:SSPM0 *(Synchronous serial port mode select bits)*: Bits de selección del modo de funcionamiento del módulo I²C

1111 = I²C esclavo, 10-bit con interrupciones Start & Stop

1110 = I²C esclavo, 7-bit cpm omterr`copmes Start & Stop

1011 = I²C maestro controlado por Firmware

1000 = I²C maestro, clock = Fosc/(4*(SSPADD+1))

0111 = I²C esclavo, direcciones 10-bit

0110 = I²C esclavo, direcciones 7-bit

El resto de combinaciones están reservadas o se utilizan únicamente en modo SPI

A2.19. SSPCON2

MODO I²C

Registro SSPCON2 (dirección 91h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
bit 7							bit 0

Valores de los bits después de un reset
R = el bit se puede leer
W = el bit se puede escribir
x = Valor desconocido

bit 7 GCEN ***(General call enable bit)*** – Modo esclavo.
1 = Habilita interrupciones cuando recibe 0000h (general call address).
0 = Deshabilita interrupciones al recibir 0000h.

bit 6 ACKSTAT ***(Acknowledge status bit)***
Modo transmisión maestro
1 = No se ha recibido confirmación de recepción del esclavo.
0 = Se ha recibido confirmación de recepción del esclavo.

bit 5 ACKDT ***(Acknowledge data bit)***
Modo recepción maestro
1 = No confirmar
0 = Confirmar
Valor transmitido cuando el usuario inicia una secuencia de confirmación al final de la recepción.

bit 4 ACKEN ***(Acknowledge sequence enable bit)***
Modo recepción maestro
1 = Inicia la secuencia de confirmación. Se pone a 0 al finalizar.
0 = Secuencia de confirmación desactivada.

bit 3 RCEN ***(Receive enable bit)*** - Modo maestro
1 = Habilita el modo de recepción I_2C.
0 = Recepción deshabilitada.

bit 2 PEN ***(Stop condition enable bit)*** – Modo maestro
1 = Inicia la secuencia de Stop. Se pone a '0' al finalizar.
0 = Secuencia de Stop deshabilitada.

bit 1 RSEN ***(Repeated start condition enabled bit)*** – Modo maestro
1 = Inicia la secuencia de Restart. Se pone a '0' al finalizar.
0 = Secuencia de Restart deshabilitada.

bit 0 SEN ***(Start condition enabled/stretch bit)***
Modo maestro
1 = Inicia la secuencia de Start. Se pone a '0' al finalizar.
0 = Secuencia de Start deshabilitada.
Modo esclavo
1 = Stretch de reloj habilitado en recepción y transmisión.
0 =Stretch de reloj habilitado en transmisión.

A2.20.- PALABRA DE CONFIGURACIÓN

CONFIG WORD (dirección 2007h)

R/P-1	U-0	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-0	U-0	U-0	R/P-1	R/P-1	R/P-1	R/P-1
CP	-	DEBUG	WRT1	WRT0	CPD	LVP	BOREN	-	-	PWRTEN	WDTEN	Fosc1	Fosc0
bit 13													bit 0

bit 13 CP ***(Code Protection)*:** Protección de escritura en memoria de programa
1 = Protección deshabilitada.
0 = Protección habilitada para toda la memoria.

bit 12 No implementado – '1'

bit 11 DEBUG : In-Circuit debug
1 = Deshabilitado, RB6 y RB7 son E/S digitales.
0 = Habilitado, RB6 y RB7 son utilizadas por el debugger.

bits 10-9 AWRT1:WRT0 ***(Write protection bits)*:** Protección parcial de memoria de programa.
11 = Deshabilitada.
10 = 0000h-00FFh protegida
01 = 0000h-03FFh protegida.
00 = 0000h-07FFh protegida.

bit 8 CPD ***(Code protection data)*:** Protección de memoria EEPROM
1 = Protección deshabilitada.
0 = Protección habilitada.

bit 7 LVP ***(Low-Voltage*** **Programming):** Programación a baja tensión
1 = Habilitada a través del terminal RB3/PGM.
0 = Deshabilitada, RB3 funciona como E/S digital.

bit 6 BOREN ***(Brown-out Reset Enable*** **bit):** Reset por caida de tensión
1 = Habilitado. 0 = Deshabilitado.

bits 5-4 No implementados – '1'

bit 3 PWRTEN ***(Power-up Timer Enable)*** Habilitación del reloj para encendido.
1 = Deshabilitado. 0 = Habilitado.

bit 2 PEN ***(Stop condition enable bit)*** – Modo maestro
1 = Inicia la secuencia de Stop. Se pone a '0' al finalizar.
0 = Secuencia de Stop deshabilitada.

bits 1-0 Focs1:Fosc0 ***(Oscillator Selection*** **Bits):** Bits de selección del modo de oscilador
11 = Oscilador RC. 10 = Oscilador HS.
01 = Oscilador XT. 00 = Oscilador LP.

A3. Introducción a MPLAB-IDE/MPLAB-SIM con Hi-Tech

A3.0. Instalación

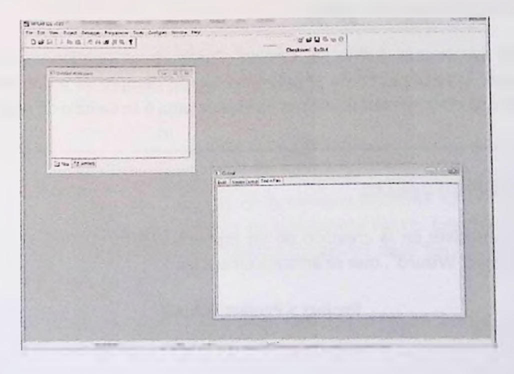
Para utilizar el entorno de programación MPLAB IDE es necesario instalar el programa, en particular se utilizará el *software* "MPLAB IDE v.8.89" que se puede descargar desde la página de microchip en la sección downloads.

http://www.microchip.com/development-tools/downloads-archive

También se utilizará el compilador de 'C' Hi-Tech, que no viene incluido en la instalación de MPLAB por lo que se deberá descargar desde:

http://www.htsoft.com/downloads/

Más concretamente, se instalará la versión "picc_9_.83_win" junto con la herramienta "UniversalToolsuite-1.37". Una vez realizada la instalación se arrancará el programa MPLAB IDE. Si no se dispone de un acceso directo en el escritorio, se puede hacer desde el menú de "Archivos de Programa" del ordenador. Una vez arrancado aparecerá la pantalla de la siguiente figura.



A3.1. Configuración

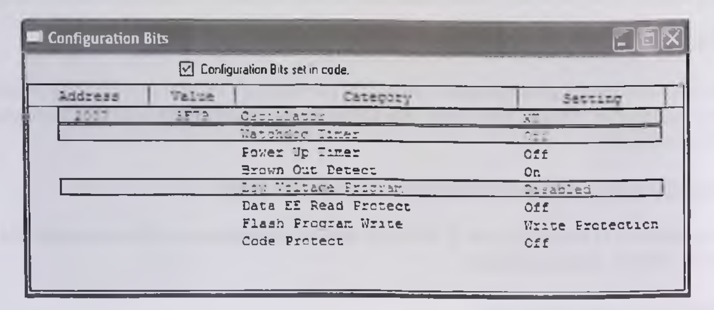
Es importante que los bits de la palabra de configuración tengan los valores adecuados en función de la aplicación. Eso se puede hacer de dos formas:

1) entrando en:

Configure > Configuration Bits

Se puede escoger el tipo de oscilador (RC, LP, XT ó HS), y activar/desactivar el watchdog, el temporizador de arranque o la protección del programa, entre otras opciones según los modelos de PIC.

Si se necesita cambiar alguna de las opciones que aparecen se deberá desactivar la casilla: "*Configuration Bits set in code*" (figura inferior), ya que si está activada solo admite como palabra de configuración la que venga especificada en el programa fuente.



2) también se puede incluir mediante la directiva "**__CONFIG**" en el programa fuente, que se encarga de que los bits de la palabra de configuración tengan los valores adecuados. Por ejemplo:

__CONFIG(WRT_OFF & WDTE_OFF & PWRTE_OFF & FOSC_XT & LVP_OFF);

que selecciona un oscilador de cristal de cuarzo, anula el watchdog, desactiva la programación a baja tensión y anula la protección del código.

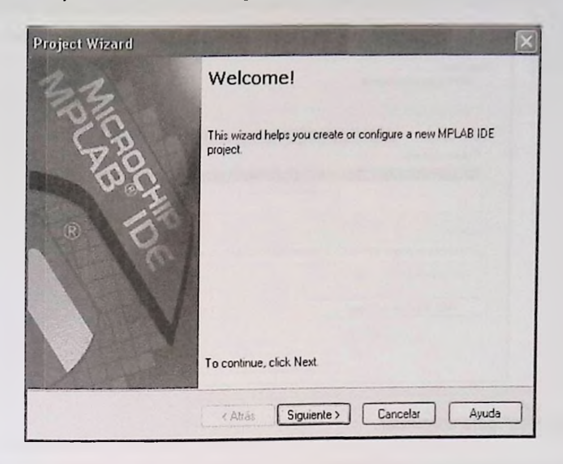
Este método tiene la ventaja frente al camino *Configure > Configuration Bits* (comentado anteriormente) de que al estar escrito en el programa fuente siempre va a determinar la palabra de configuración, aunque se realicen modificaciones en el programa o se cambie de ordenador o de ICD2.

A3.2. Creación del proyecto

1) El siguiente paso consiste en la creación de un Proyecto. La forma más sencilla de hacerlo es utilizar el "MPLAB Project Wizard", que se arranca en el menú:

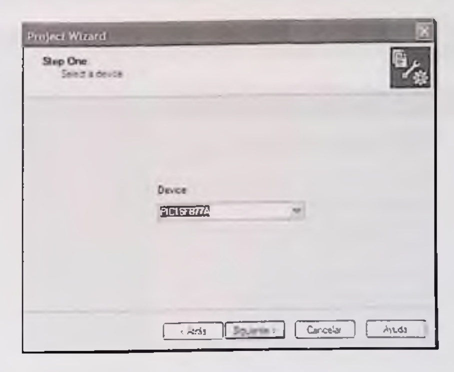
Project > Project Wizard

La pantalla toma un aspecto como el de la siguiente figura:



Seleccionar "Siguiente" para continuar.

2) Seleccionar, de la lista de dispositivos disponibles, el procesador a utilizar.

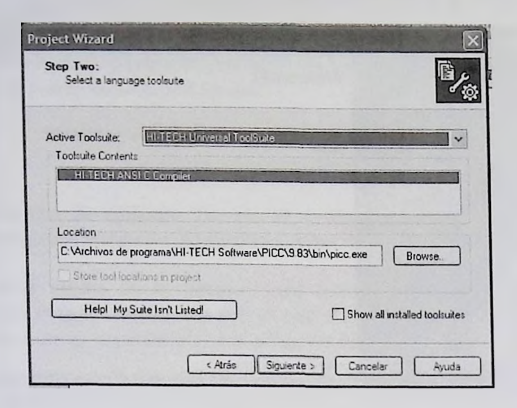


Seleccionar "Siguiente" para continuar.

3) Seleccionar la herramienta a utilizar ("*Active Toolsuite*") que debe ser Microchip **HI-TECH Universal ToolSuite**. Al hacerlo, nos aparecen en la ventana siguiente las herramientas disponibles *(Toolsuite Contents)*, como se puede observar en la siguiente figura. Esas herramientas deben ser: *HI-TECH ANSI C Compiler*. Y en la ventana siguiente (*Location*) debe figurar la trayectoria completa de esos tres programas ejecutables:

C:\Archivos de programa\HI-TECH Softare\picc\9.83\bin\picc.exe

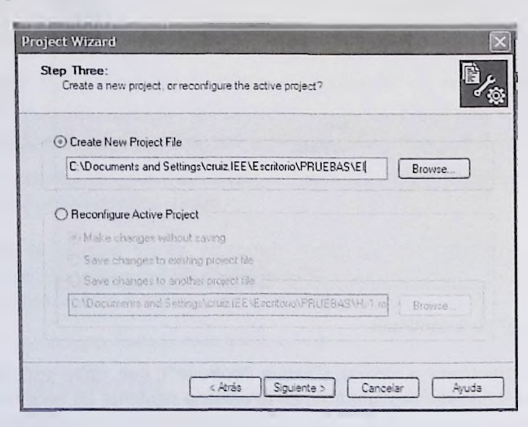
Si esa trayectoria está incompleta o es errónea se debe pulsar "*Browse*" para localizarlo.



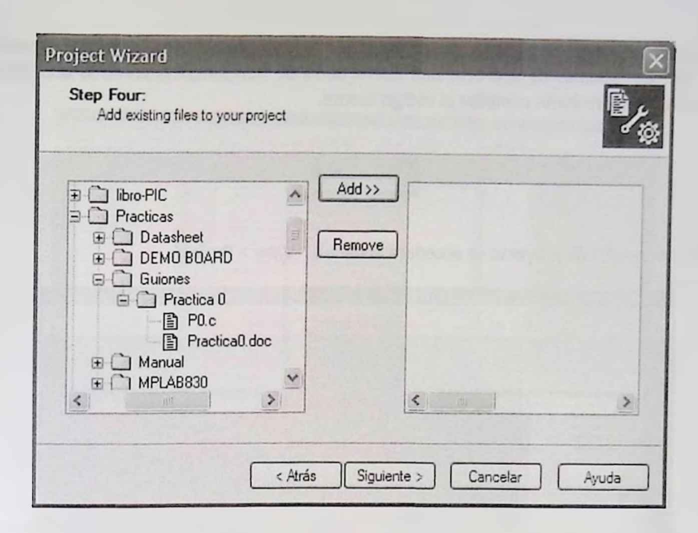
Seleccionar "Siguiente" para continuar.

4) El siguiente paso es asignarle un nombre al proyecto. Debe estar en el mismo directorio donde se encuentres los programas con el código fuente en caso de que ya existan y es conveniente darle el mismo nombre para localizar mejor todo lo concerniente a un mismo ejercicio.

Para eso, en la ventana "*Create New Project File*", pulsar en "Browse" y acceder al subdirectorio en el que se esté trabajando.

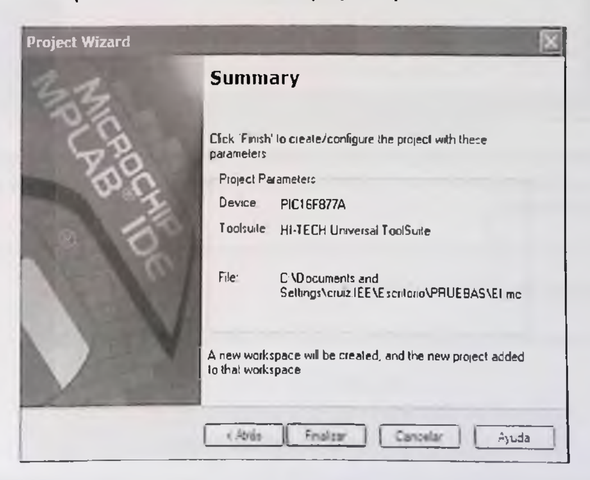


5) La pantalla siguiente pregunta qué ficheros se van a incorporar al proyecto. En el caso de que ya existan se deben seleccionar y añadir (pulsar en "*Add*"). Es conveniente que en la ventana de la derecha y a la izquierda de la trayectoria figure una "A" (indica automático).



Seleccionar "Siguiente" para continuar.

Aparece entonces una pantalla con un resumen del proyecto que se va a crear:

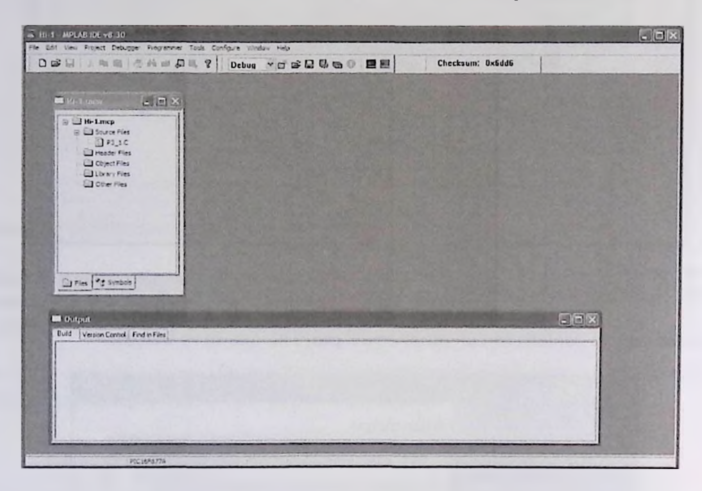


Si esos datos no son correctos se deberá pulsar "Atrás" y corregir donde sea oportuno. Si son correctos, se puede pulsar en "Finalizar" con lo que se terminará de crear el proyecto y se saldrá del "*Project Wizard*".

Una vez creado el proyecto se abrirá MPLAB junto con las ventanas de navegador del proyecto y la ventana "*Output*" además de aparecer una nueva barra de herramientas como la que aparece a continuación, que permitirán compilar el código fuente.



Para ver el navegador de proyecto se accederá al menú: ***View > Project***



Se pueden añadir archivos y salvar proyectos pulsando el botón derecho del ratón desde la ventana de proyecto. Los ficheros también se pueden borrar manualmente seleccionándolos y utilizando el botón derecho del ratón.

A3.3. Creación del fichero fuente

Para generar el programa fuente se utiliza el editor del MPLAB IDE. Seleccionando:

File > New

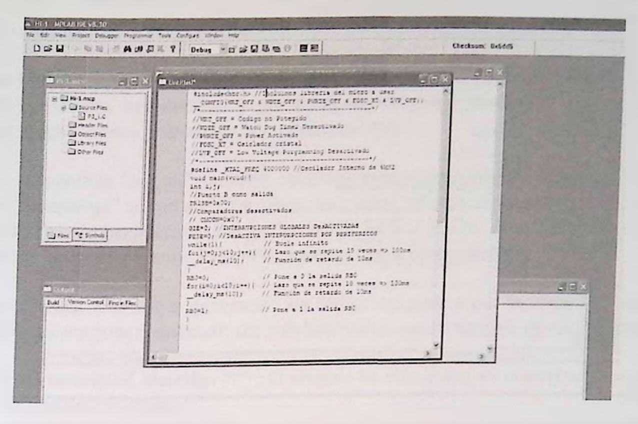
o

File > Add New File to Project

aparecerá en el área de trabajo una ventana en blanco en la que va a quedar escrito el programa fuente.

Para crear el programa se puede utilizar cualquier editor de texto en caracteres ASCII y hay que tener en cuenta las <u>normas programación en C</u>. descritas en el primer capítulo.

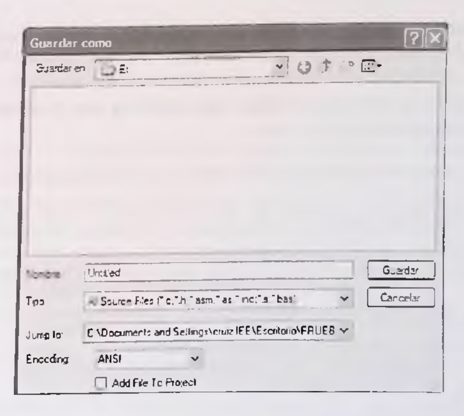
En la siguiente figura se puede observar que el fichero no tiene todavía un nombre (untitled) y que todo el texto tiene el mismo color.



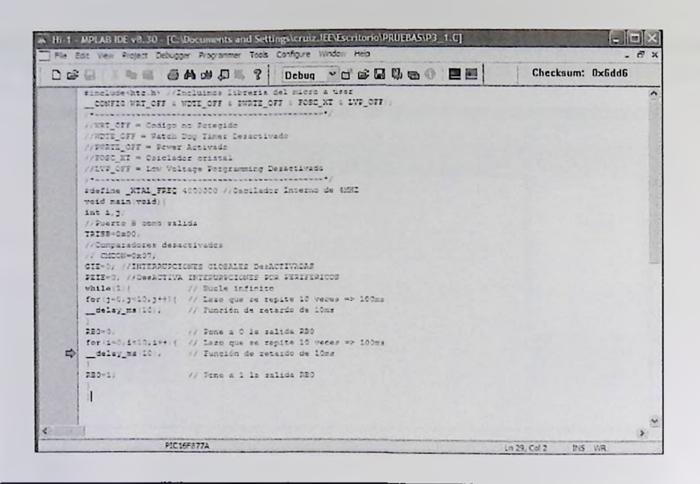
Es conveniente ir guardando el texto a medida que se va escribiendo:

File > Save as
ó: **File > Save**

Al guardar el fichero debe tener la extensión ".c". Para eso hay que tener cuidado con el formato y en la ventana "tipo" debe estar seleccionada la opción: "**All Source Files**".



Tras guardar el programa fuente, el texto aparece en diferentes colores, que diferencian las instrucciones, los comentarios, constantes, etc, como se puede ver a continuación. Esos colores se pueden configurar a gusto del usuario. Para más información acudir a: *Help > MPLAB Editor Help*.



A3.4. Compilación del proyecto

Una vez que el proyecto está creado y se cuenta con un archivo fuente es necesario compilarlo o lo que es lo mismo, generar el código máquina. Para eso el entorno MPLAB utiliza el **HI-TECH ANSI C compiler**. El procedimiento a seguir es el siguiente:

Project > Build All

También puede utilizarse el método abreviado de teclado F10 o el icono correspondiente de la barra de herramientas.

El caso más común cuando se empieza a trabajar con MPLAB es que, tras intentar compilar, en la pantalla aparezca un mensaje semejante a este:

```
Build C:\Documents and Settings\cruiz.IEE\Escritorio\PRUEBAS\Hi-1 for device
16F877A
Using driver C:\Archivos de programa\HI-TECH Software\PICC\9.83\bin\picc.exe
Make: The target "C:\Documents and Settings\cruiz.IEE\Escritorio\PRUEBAS\P3_1.p1"
is out of date.
Executing: "C:\Archivos de programa\HI-TECH Software\PICC\9.83\bin\picc.exe" --
pass1 "C:\Documents and Settings\cruiz.IEE\Escritorio\PRUEBAS\P3_1.C" -q --
chip=16F877A -P --runtime=default,+clear,+init,-keep,+osccal,-download,-
resetbits,-stackcall,+clib --opt=default,+asm,-debug,-speed,+space,9 --warn=0 -
D__DEBUG=1 --double=24 --float=24 --addrqual=ignore -g --asmlist "--
errformat=Error   [%n] %f; %l.%c %s" "--msgformat=Advisory[%n] %s" "--
warnformat=Warning [%n] %f; %l.%c %s"
Error [192] C:\Documents and Settings\cruiz.IEE\Escritorio\PRUEBAS\P3_1.C; 14.1
undefined identifier "TRIS"
Error [192] C:\Documents and Settings\cruiz.IEE\Escritorio\PRUEBAS\P3_1.C; 24.5
undefined identifier "i"
Error [312] C:\Documents and Settings\cruiz.IEE\Escritorio\PRUEBAS\P3_1.C; 26.1
";" expected
Error [195] C:\Documents and Settings\cruiz.IEE\Escritorio\PRUEBAS\P3_1.C; 29.0
```

```
expression syntax
Error [300] C:\Documents and Settings\cruiz.IEE\Escritorio\PRUEBAS\P3_1.C; 29.0
unexpected end of file

********** Build failed! **********
```

En la última línea se puede leer: "*BUILD FAILED*", es decir, el compilador **HI-TECH** no ha sido capaz de generar un fichero hexadecimal y por tanto no se podrá simular el comportamiento del programa ni mucho menos grabarlo en la memoria del PIC.

Además, en la pantalla "*output*" también podemos tener disponible otras informaciones, como "***messages***", "***warnings***" y "***errors***". De estos tres tipos, el más importante, porque impide la generación del fichero hexadecimal son los "*errors*", mientras que los otros dos tipos no impiden el ensamblado del programa fuente. Pasemos a comentarlos a continuación:

- **Errores (Error).** Impiden la generación de código máquina y por lo tanto es imprescindible corregirlos para poder continuar. Los más habituales, con su número de caracterización, son:

  - **(192) undefined identifier "*"** : El símbolo ha sido usado en el programa pero no se ha definido previamente.
  - **(195) expression syntax** : Error de sintaxis que indica que la expresión está mal formada.
  - **(300) unexpected end of file:** Final de código inesperado que indica que falta cerrar la última llave.
  - **(312) "*" expected:** Se esperaba el argumento indicado entre comillas.

Es importante tener en cuenta que la eliminación de estos errores simplemente permite obtener un fichero ejecutable, pero no aporta ninguna información acerca de si el programa funcionará o no correctamente.

- **Advertencias (Warning).** Estos mensajes no impiden la obtención del fichero hexadecimal, pero advierten de algo que al programa ensamblador le parece extraño. Es conveniente comprobarlos todos.

Para tratar los mensajes, el camino más rápido es **hacer doble clic en la línea del fichero "*output*" en la que está el mensaje.** Eso hace que el cursor se ponga en la línea del programa fuente que da lugar a la aparición de ese mensaje permitiendo corregirlo. A continuación, el fichero fuente se graba de nuevo, "***File > Save***", y se vuelve a ensamblar: "***Project > Build All***". El proceso se repite hasta que estén corregidos todos, momento en el que **HI-TECH** consigue generar el fichero hexadecimal, apareciendo una pantalla donde se puede leer "***BUILD SUCCEEDED***".

```
Make: The target "C:\Documents and Settings\cruiz.IEE\Escritorio\PRUEBAS\P3_1.p1"
is out of date.
Executing: "C:\Archivos de programa\HI-TECH Software\PICC\9.83\bin\picc.exe" --
pass1 "C:\Documents and Settings\cruiz.IEE\Escritorio\PRUEBAS\P3_1.C" -q --
chip=16F877A -P --runtime=default,+clear,+init,-keep,+osccal,-download,-
resetbits,-stackcall,+clib --opt=default,+asm,-debug,-speed,+space,9 --warn=0 -
D__DEBUG=1 --double=24 --float=24 --addrqual=ignore -g --asmlist "--
errformat=Error [%n] %f; %l.%c %s" "--msgformat=Advisory[%n] %s" "--
warnformat=Warning [%n] %f; %l.%c %s"
Executing: "C:\Archivos de programa\HI-TECH Software\PICC\9.83\bin\picc.exe" -
oHi-1.cof -mHi-1.map --summary=default,-psect,-class,+mem,-hex --output=default,-
inhx032 P3_1.p1 --chip=16F877A -P --runtime=default,+clear,+init,-keep,+osccal,-
```

```
download,-resetbits,-stackcall,+clib --opt=default,+asm,-debug,-speed,+space,9 --
warn=0 -D__DEBUG=1 --double=24 --float=24 --addrqual=ignore -g --asmlist "--
errformat=Error [%n] %f; %l.%c %s" "--msgformat=Advisory[%n] %s" "--
warnformat=Warning [%n] %f; %l.%c %s"
HI-TECH C Compiler for PIC10/12/16 MCUs (Lite Mode) V9.83
Copyright (C) 2011 Microchip Technology Inc.
(1273) Omniscient Code Generation not available in Lite mode (warning)

Memory Summary:
    Program space        used    71h (   113) of  2000h words    (  1.4%)
    Data space           used     8h (     8) of   170h bytes    (  2.2%)
    EEPROM space         used     0h (     0) of   100h bytes    (  0.0%)
    Configuration bits   used     1h (     1) of     1h word     (100.0%)
    ID Location space    used     0h (     0) of     4h bytes    (  0.0%)

Running this compiler in PRO mode, with Omniscient Code Generation enabled,
produces code which is typically 40% smaller than in Lite mode.
See http://microchip.htsoft.com/portal/pic_pro for more information.

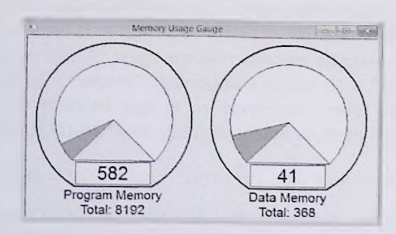
Loaded C:\Documents and Settings\cruiz.IEE\Escritorio\PRUEBAS\Hi-1.cof.

********** Build successful! **********
```

La pantalla output también informa de:

- procesador utilizado (en el ejemplo, 16F877A)
- nombre del proyecto y su trayectoria (en el ejemplo, P3_1.C)
- herramienta utilizada: picc.exe, con su trayectoria completa
- fecha y hora en la que se produjo la compilación
- Espacio utilizado en memoria de programa, memoria de datos, etc.

Este último apartado que hace referencia a la utilización de la memoria del PIC cobra especial importancia a la hora de desarrollar aplicaciones complejas ya que será un indicador de la necesidad de migrar hacia un microcontrolador con mayores prestaciones. Esta información también está accesible de forma gráfica a través del menú *View|Memory Usage Gauge* como se muestra en la siguiente figura.



A partir de este momento, ya se dispone de un programa ejecutable y se puede simular el funcionamiento o grabarlo en la memoria permanente del microcontrolador. El código máquina ejecutable del programa escrito en ensamblador se encuentra en el archivo .hex con el mismo nombre que el archivo .c que se ha compilado. Este archivo ".hex" contiene además información relativa a la grabación del mismo en las diferentes posiciones de memoria de programa del microcontrolador. Se puede observar cómo se ubicaría dicho programa en la memoria del

microcontrolador accediendo al menú *View/Program Memory* que en las pestañas inferiores permitirá elegir entre la representación en hexadecimal de los datos como aparece en la siguiente figura o una representación más cercana al lenguaje máquina (lenguaje ensamblador).

Program Memory

Address									ASCII
0000	120A	118A	2812	3FFF	00FE	0E03	1283	1303	.....(.? ....
0008	0080	0804	0081	080A	0082	087F	0083	120A	....
0010	118A	2953	120A	118A	2A42	1283	1303	00A7	..S).... B*......
0018	3020	00A2	0822	00A5	28EE	082D	3A25	1903	0.."... .[illegible]..
0020	2822	2823	282A	082D	120A	118A	2231	120A	"(#(*(-. ....1"..
0028	118A	28EE	01A6	2831	2904	2846	2846	28EE	...(..1(.)F(F(.(
0030	2846	3001	07A7	30FF	0727	0084	120A	118A	F(.0...0 '.......
0038	2208	120A	118A	00AD	3A00	1903	2904	3A64	."...... .:...)d:
0040	1903	2846	3A0D	1903	2846	28EE	0825	0084	..F(.:.. F(.(%...
0048	1383	0800	00A9	0A84	0800	00AA	3002	00A2	....0..

Opcode Hex | Machine | Symbolic

A3.5. Simulación del proyecto

Llegados a este punto es necesario comprobar el funcionamiento del programa, es decir, simular su funcionamiento.

Esta simulación se puede realizar utilizando el simulador por software **MPLAB-SIM** que viene incluido con **MPLAB IDE**, como se explicará a continuación.

MPLAB-SIM es un simulador para los microcontroladores PIC que viene integrado en el entorno MPLAB IDE. Permite modificar el programa y ejecutarlo a continuación, introducir estímulos externos y observar la ejecución del programa objeto.

La velocidad de ejecución, aunque llega siempre a la máxima posible, es varios órdenes de magnitud más baja que la del procesador real y depende del ordenador y de otros factores. Puede llegar al orden de unos ms por instrucción (un PIC real, con un cristal de 4 MHz, emplea 1 µs por instrucción, salvo si es de salto, que emplea 2 µs).

Para arrancar el simulador es necesario indicar que se va a utilizar la herramienta MPLAB SIM y para ello se accederá al menú:

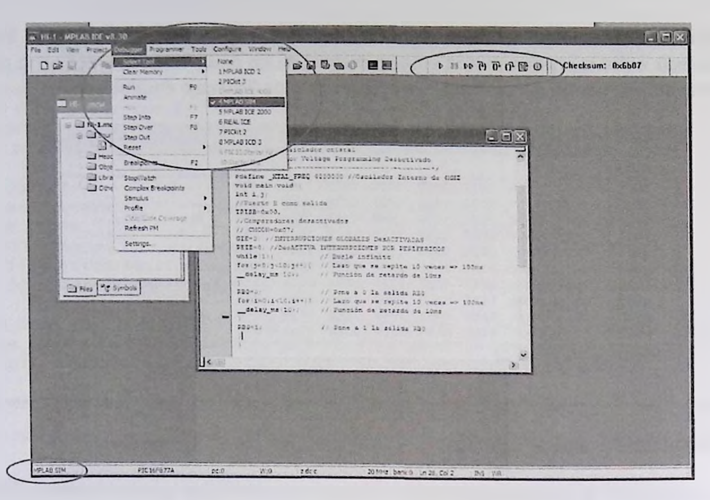
Debugger > Select Tool > MPLAB SIM

A continuación se producen los siguientes cambios:

1) La ventana de la izquierda de la barra de estado (parte inferior de la pantalla) cambia a MPLAB SIM

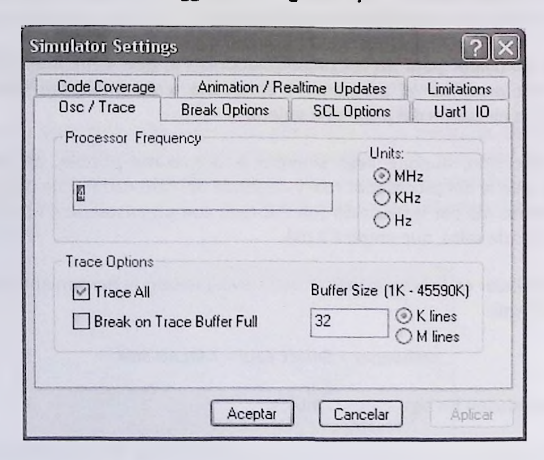
2) Aparecen nuevas opciones en el menú "*debugger*"

3) En la barra de herramientas aparecen los iconos correspondientes al simulador



Nuestro programa está listo para ser ejecutado. Debemos introducir ahora la frecuencia que va a tener el oscilador.

Debugger > Settings > Osc/Trace



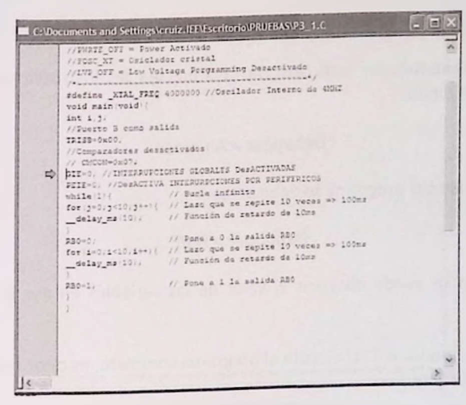
En primer lugar, es conveniente que el simulador empiece por ejecutar la primera instrucción del programa, para eso se debe realizar un "*reset*" del procesador:

Debugger > Reset > Processor reset

O también se puede actuar sobre el teclado o sobre el icono correspondiente de la siguiente lista:

Debugger Menu	Botones	Hot Key
RUN		F9
HALT		F5
ANIMATE		
STEP INTO		F7
STEP OVER		F8
STEP OUT OF		
RESET		F6
BREAKPOINT		F2

En la siguiente pantalla aparece una flecha verde en el margen izquierdo de la ventana donde está escrito el programa fuente. La flecha apuntará siempre a la primera instrucción que se va a ejecutar en cuanto se dé la orden de ejecución.



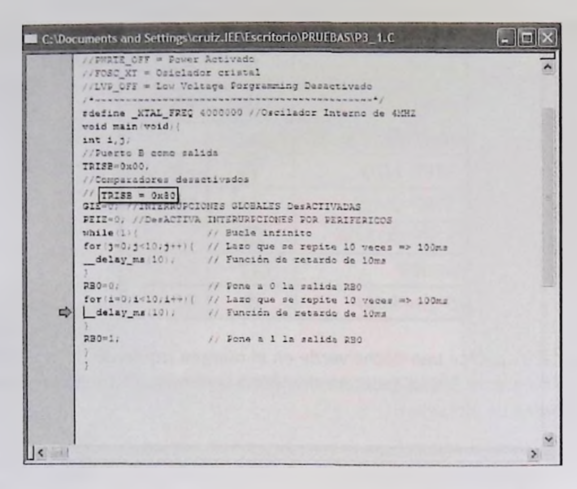
A partir de aquí existen tres posibilidades de ejecutar el programa: **paso a paso, en modo animado y total**.

1) Paso a paso (Step). En esta modalidad, la CPU ejecuta las instrucciones una a una cada vez que se acceda al menú:

Debugger > Step Into

O también actuando sobre la tecla F7 o sobre el icono correspondiente.

Ejecutando el programa de esta manera es posible observar el valor de las variables colocando el cursor sobre ellas.



2) Modo animado (Animate). En este caso, la CPU ejecuta las instrucciones una tras otra sin esperar. Se activa en el menú:

Debugger > Animate

Para detener la ejecución del programa se debe actuar en:

Debugger > Halt

También en este caso se puede observar el valor de las variables en ese instante colocando el cursor sobre ellas.

3) Total (Run). En este modo, la CPU ejecuta el programa completo, es decir desde la primera hasta la última instrucción. Se activa en:

Debugger > Run

O también actuando sobre la tecla F9 o sobre el icono correspondiente.

En barra de estado aparece la palabra "*running*".

Para detener la ejecución del programa se debe acceder al menú:

Debugger > Halt

O también mediante la tecla F5 o sobre el icono correspondiente.

En este caso, si al colocar el cursor sobre una variable no aparecerá el valor que tiene en ese instante. Para observarlo, primero se deberá detener la ejecución del programa.

A3.6. Observando el funcionamiento

Hay varias formas de observar lo que sucede al ir ejecutándose las instrucciones del programa. Dentro del menú "**View**" existen varias posibilidades:

Project:	resumen del proyecto
***Output*:**	el fichero de salida, que ya conocemos
***Disassembly Listing*:**	listado del programa fuente (sin ensamblar)
***EEPROM*:**	estado de la memoria EEPROM
***File Registers*:**	valores almacenados en los registros
***Hardware Stack*:**	estado de la pila
***Program Memory*:**	estado de la memoria de programa, con las instrucciones
Special Function Registers:	estado de los registros de funciones especiales
***Simulator Trace*:**	almacena todos los pasos del simulador

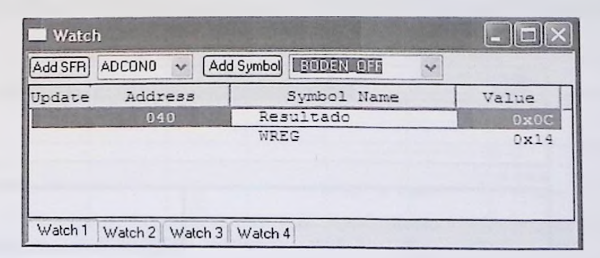
Cualquiera de esas posibilidades proporciona información acerca del estado del microcontrolador durante la ejecución del programa. El inconveniente es que tanto el contenido de los registros como su dirección pueden estar en hexadecimal por lo que es bastante engorroso. Además, hay que tener en cuenta que en programas complejos pueden ser muchos los valores que cambian con cada instrucción, lo que dificulta el seguimiento de unas pocas variables.

A3.6.1. Visualización de variables

Para evitar este problema se puede abrir una "ventana de observación":

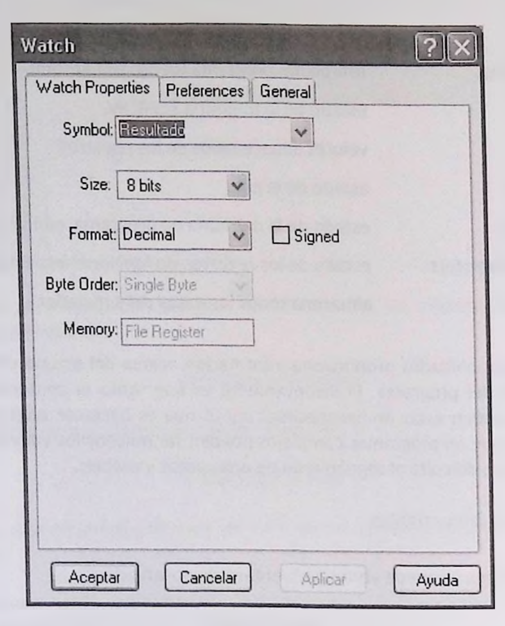
View > Watch

Con eso aparecerá una nueva ventana como se observa en la siguiente figura en la que se pueden seleccionar tanto los registros de funciones especiales (SFR) como los símbolos (variables) que se quieren visualizar.

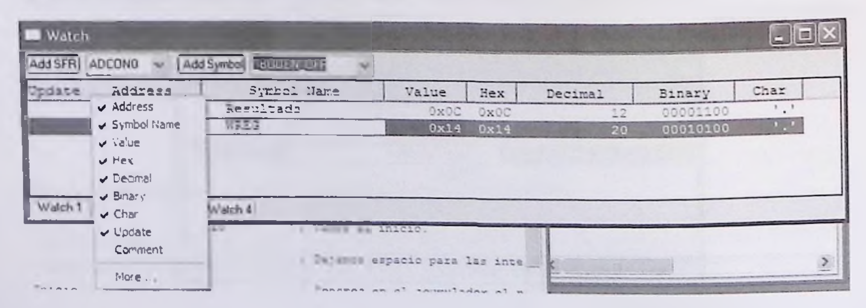


Una vez que se tiene en la ventana el símbolo se puede escoger el formato en el que aparecerá representado seleccionándolo y apretando el botón derecho del ratón para entrar en sus propiedades. Ahí se podrán escoger varias opciones:

Symbol: símbolo
Size: número de bits a observar
Format: formato (aquí podemos escoger también un solo bit, binario, decimal, ascii...)



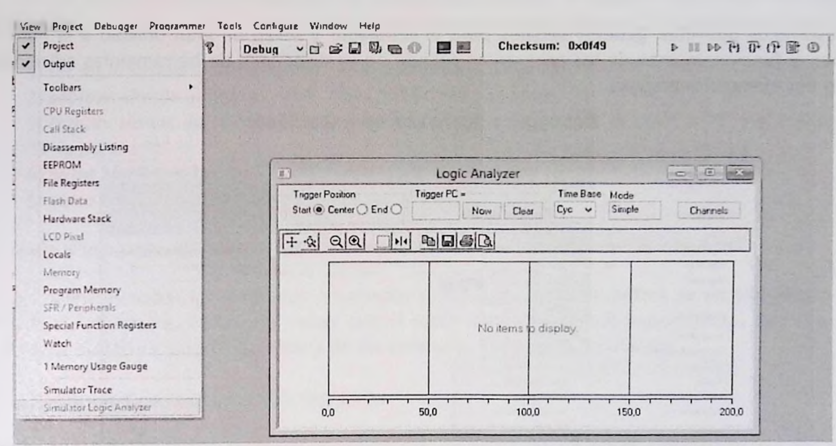
También se puede situar el cursor sobre la barra "*address*" y actuar sobre el botón derecho del ratón. Eso permitirá escoger entre diferentes formatos de presentación como se muestra en la figura inferior.



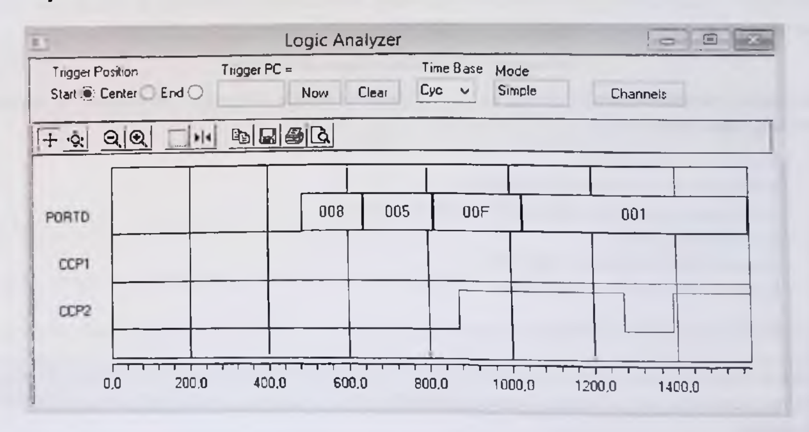
A3.6.2. Visualización de señales

Además de visualizar el valor que se almacena en cada uno de los registros o variables del programa a veces es conveniente ver cómo evolucionan durante la ejecución del mismo. Para eso se puede utilizar el visualizador de señales "*Simulator Logic Analyzer*" que se encuentra en el menú *View*.

View > Simulator Logic Analyzer



Esta utilidad permite mostrar el valor que han ido tomando una o varias señales cuando se detiene la ejecución del programa o cuando este se ejecuta en modo animado. Para ello basta con pulsar el botón *Channels* y elegir el terminal que se desea visualizar o bien agrupar varios terminales de forma conjunta formando un bus de datos.

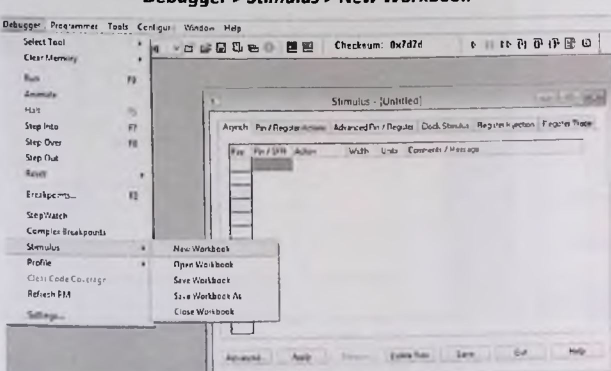


A3.7. Generador de estímulos

El simulador MPLAB SIM dispone también de una herramienta que permite simular estímulos externos tales como pulsadores, entradas de reloj etc. Este simulador evalúa los estímulos y genera todas las respuestas en los límites de cada ciclo de instrucción (Tcy = 4·Tosc). Por ese motivo, algunos sucesos físicos no pueden simularse con precisión, en particular los sucesos puramente asíncronos y los sucesos de periodo más corto que un ciclo de instrucción.

Los estímulos permiten generar señales para el simulador y serán de gran utilidad a la hora de probar el funcionamiento de los programas cuando no se dispongan de herramientas adicionales como por ejemplo debuggers.

Debugger > Stimulus > New Workbook



También existe la opción de guardar el fichero de estímulos. Basta con actuar sobre *Save*, darle el mismo nombre que al resto del proyecto y guardarlo en el mismo subdirectorio. Posteriormente se puede recuperar actuando sobre:

Debugger > Stimulus > Open Workbook

Los estímulos permiten poner entradas a "0" o a "1" y cargar valores directamente en registros. Existen seis tipos diferentes:

- Estímulos asíncronos
- Estímulos de terminales o registros
- Estímulos avanzados sobre terminales o registros
- Estímulos de reloj
- Inserción de valores en registros
- Captura de valores de registros

A3.7.1. Estímulos Asíncronos

Los estímulos asíncronos permiten simular +5 V y 0 V en terminales configurados como entradas. Se activan haciendo clic en el botón correspondiente de la ventana de diálogo. Para definirlos se accede desde:

Activando la pestaña "*Asynch*" se pueden definir los estímulos.

La primera columna (*fire)* es la que se tiene que pulsar para que se ejecute la simulación del estímulo que esté configurando a su derecha.

La segunda columna (*Pin/SFR*) permite escoger la patilla/terminal en la que se generará el estímulo. Basta con pulsar dentro del cuadro en blanco para que aparezcan las opciones.

En la tercera columna (Action) se escogerá la acción a simular de entre las cinco posibilidades:

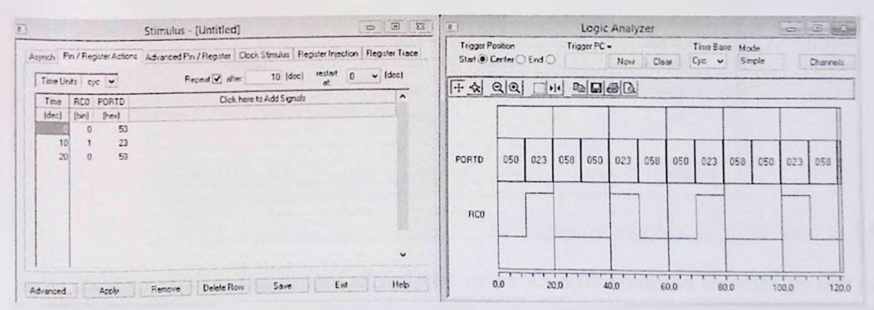
1) ***set high:*** simula una tensión "1" en la patilla seleccionada
2) ***set low:*** simula una tensión "0" en la patilla seleccionada
3) ***toggle:*** simula un cambio, es decir, si esa patilla estaba a "1" la pone a "0" y si estaba a "0" la pone a "1"
4) ***pulse high:*** simula un pulso en "1" de corta duración
5) ***pulse low:*** simula un pulso en "0" de corta duración

Si se deséa eliminar un estímulo, basta con seleccionar la línea y pulsar en "***Delete Row***".

Una vez definidos todos los estímulos necesarios pueden activarse mientras se va simulando la ejecución del programa. Basta con pulsar con el ratón en el botón correspondiente, por eso es conveniente mantener abierta la ventana de los estímulos durante la simulación.

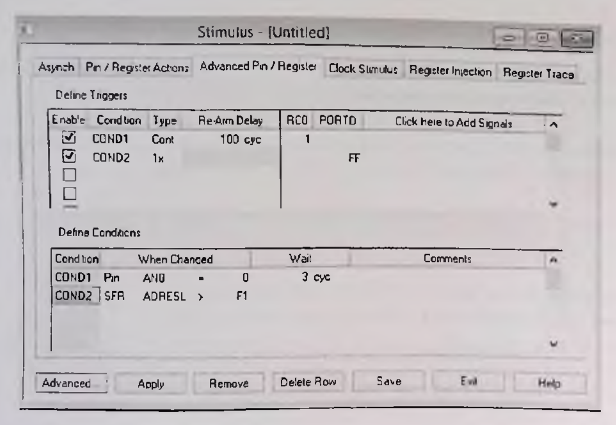
A3.7.2. Estímulos de terminales o registros

Los estímulos de terminales o registros permiten introducir valores en terminales o registros de forma automática durante el tiempo definido además de poder repetir la introducción de forma cíclica seleccionando la casilla *repeat*.



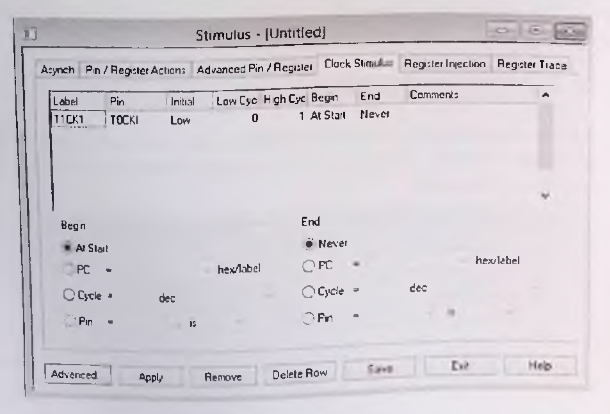
A3.7.3. Estímulos avanzados de terminales o registros

Los estímulos avanzados de terminales o registros permiten introducir valores en terminales o registros de forma automática cuando se den determinadas condiciones predefinidas como que un terminal o registro tomen o superen un determinado valor.



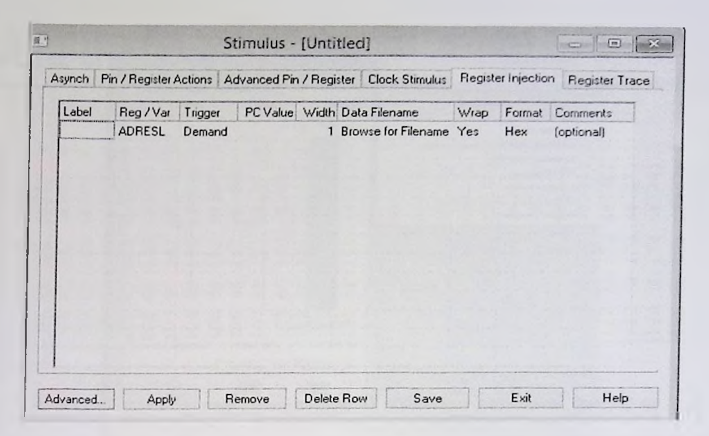
A3.7.4. Estímulos de reloj

Los estímulos de reloj permiten la generación de una señal periódica en cualquiera de los terminales así como definir el momento en el que comenzará y terminará esta señal.



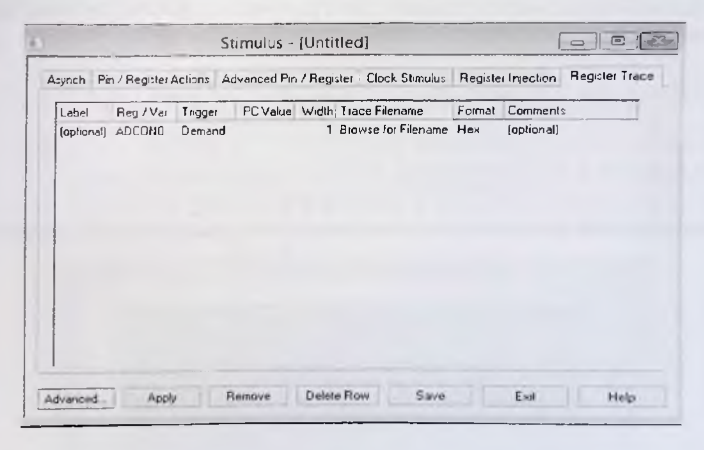
A3.7.5. Inserción de valores en registros

Este tipo de estímulo permite introducir en los registros los valores que se hayan colocado en un archivo externo. Estos estímulos serán útiles cuando se quiera simular el funcionamiento de por ejemplo los módulos de comunicaciones USART o el conversor analógico digital, ya que el microcontrolador irá recogiendo cada uno de los valores almacenados en el archivo de texto cada vez que acceda al registro asociado a estos módulos.



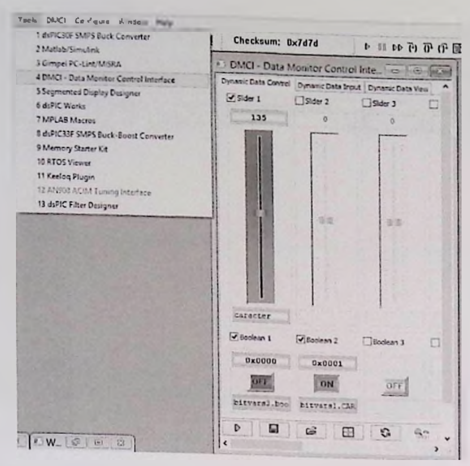
A3.7.6. Captura de valores de registros

Esta opción permite almacenar en un archivo de texto los valores que van tomando los registros, permitiendo seleccionar el momento de comienzo y el formato en el que se almacenarán.

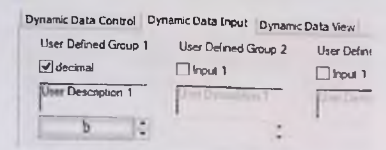


A3.7. Interfaz de control

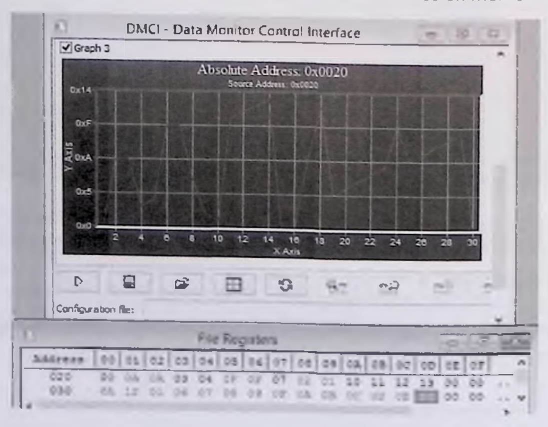
Otra de las utilidades que presenta MPLABSIM es el "*Data Monitor Control Interface*" a la que se accede desde el menú *Tools -> Data Monitor Control Interface*.



Mediante esta herramienta es posible modificar los valores que van tomando las variables y registros durante la ejecución del programa utilizando controles como barras de desplazamiento o botones que aparecen en la pestaña "*Dynamic Data Control*" o los cuadros de control en la pestaña "*Dynamic Data Input*".



Esta herramienta también permite visualizar los valores almacenados en memoria de forma gráfica.



Dec	Hx	Oct	Char	
0	0	000	NUL	(null)
1	1	001	SOH	(start of heading)
2	2	002	STX	(start of text)
3	3	003	ETX	(end of text)
4	4	004	EOT	(end of transmission)
5	5	005	ENQ	(enquiry)
6	6	006	ACK	(acknowledge)
7	7	007	BEL	(bell)
8	8	010	BS	(backspace)
9	9	011	TAB	(horizontal tab)
10	A	012	LF	(NL line feed, new line)
11	B	013	VT	(vertical tab)
12	C	014	FF	(NP form feed, new page)
13	D	015	CR	(carriage return)
14	E	016	SO	(shift out)
15	F	017	SI	(shift in)
16	10	020	DLE	(data link escape)
17	11	021	DC1	(device control 1)
18	12	022	DC2	(device control 2)
19	13	023	DC3	(device control 3)
20	14	024	DC4	(device control 4)
21	15	025	NAK	(negative acknowledge)
22	16	026	SYN	(synchronous idle)
23	17	027	ETB	(end of trans. block)
24	18	030	CAN	(cancel)
25	19	031	EM	(end of medium)
26	1A	032	SUB	(substitute)
27	1B	033	ESC	(escape)
28	1C	034	FS	(file separator)
29	1D	035	GS	(group separator)
30	1E	036	RS	(record separator)
31	1F	037	US	(unit separator)

Dec	Hx	Oct	Html	Chr
32	20	040		Space
33	21	041	!	!
34	22	042	"	"
35	23	043	#	#
36	24	044	$	$
37	25	045	%	%
38	26	046	&	&
39	27	047	'	'
40	28	050	(	(
41	29	051	)	)
42	2A	052	*	*
43	2B	053	+	+
44	2C	054	,	,
45	2D	055	-	-
46	2E	056	.	.
47	2F	057	/	/
48	30	060	0	0
49	31	061	1	1
50	32	062	2	2
51	33	063	3	3
52	34	064	4	4
53	35	065	5	5
54	36	066	6	6
55	37	067	7	7
56	38	070	8	8
57	39	071	9	9
58	3A	072	:	:
59	3B	073	;	;
60	3C	074	<	<
61	3D	075	=	=
62	3E	076	>	>
63	3F	077	?	?

A4. Tabla de códigos ASCII

Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
64	40	100	@	@	96	60	140	`	`
65	41	101	A	A	97	61	141	a	a
66	42	102	B	B	98	62	142	b	b
67	43	103	C	C	99	63	143	c	c
68	44	104	D	D	100	64	144	d	d
69	45	105	E	E	101	65	145	e	e
70	46	106	F	F	102	66	146	f	f
71	47	107	G	G	103	67	147	g	g
72	48	110	H	H	104	68	150	h	h
73	49	111	I	I	105	69	151	i	i
74	4A	112	J	J	106	6A	152	j	j
75	4B	113	K	K	107	6B	153	k	k
76	4C	114	L	L	108	6C	154	l	l
77	4D	115	M	M	109	6D	155	m	m
78	4E	116	N	N	110	6E	156	n	n
79	4F	117	O	O	111	6F	157	o	o
80	50	120	P	P	112	70	160	p	p
81	51	121	Q	Q	113	71	161	q	q
82	52	122	R	R	114	72	162	r	r
83	53	123	S	S	115	73	163	s	s
84	54	124	T	T	116	74	164	t	t
85	55	125	U	U	117	75	165	u	u
86	56	126	V	V	118	76	166	v	v
87	57	127	W	W	119	77	167	w	w
88	58	130	X	X	120	78	170	x	x
89	59	131	Y	Y	121	79	171	y	y
90	5A	132	Z	Z	122	7A	172	z	z
91	5B	133	[	[	123	7B	173	{	{
92	5C	134	\	\	124	7C	174			|
93	5D	135	]	]	125	7D	175	}	}
94	5E	136	^	^	126	7E	176	~	~
95	5F	137	_	_	127	7F	177	&#127;	DEL



MICROCONTROLADORES

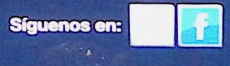
La utilización de microcontroladores permite resolver de manera sencilla gran cantidad de problemas electrónicos cotidianos asociados con el control de dispositivos sensores y actuadores. El conocimiento del funcionamiento y la programación de estos sistemas embebidos integran una parte fundamental del currículo para los estudiantes de ingeniería y les proporciona una visión actual de la tecnología utilizada en la industria y les prepara para la realización de futuros proyectos basados en sistemas embebidos.

Este libro, elaborado por tres profesores del área de Tecnología Electrónica de la Universidad Pública de Navarra, está pensado para servir de introducción y apoyo práctico a todos aquellos estudiantes de ingeniería y personas en general interesadas en el diseño electrónico mediante la utilización de sistemas microcontroladores de 8 bits. Así, el libro se estructura en diferentes capítulos en los que mediante la realización de ejercicios de diferente complejidad se revisan los módulos que se encuentran generalmente integrados en estos sistemas, como por ejemplo los puertos de entrada/salida digitales, temporizadores, conversores analógicos, comparadores y comunicaciones serie.

Los ejemplos y ejercicios del [illegible]dos en la arquitectura del PIC16F877A pero se pueden extrapolar fácil[illegible] microcontroladores gracias al empleo de la programación en lenguaj[illegible]. No obstante, las estructuras básicas de programación de este lenguaje [illegible] el primer capítulo del libro para aquellos menos familiarizados con el mismo [illegible] los capítulos siguientes están orientados a la resolución de problemas prác[illegible]

La metodología empleada permite [illegible]emplos propuestos a partir de los conocimientos básicos de programación de los p[illegible] capítulos y sin la necesidad de utilizar hardware ni elementos adicionales. Todo ello gracias a la utilización de la herramienta de software gratuita MPLAB IDE®, que integra un simulador, y a la descarga de los materiales necesarios de forma online en la sección de descargas del libro en www.marcombo.com





www.marcombo.com



MICROCONTROLADORES

La utilización de microcontroladores permite resolver de manera sencilla gran cantidad de problemas electrónicos cotidianos asociados con el control de dispositivos sensores y actuadores. El conocimiento del funcionamiento y la programación de estos sistemas embebidos integran una parte fundamental del currículo para los estudiantes de ingeniería y les proporciona una visión actual de la tecnología utilizada en la industria y les prepara para la realización de futuros proyectos basados en sistemas embebidos.

Este libro, elaborado por tres profesores del área de Tecnología Electrónica de la Universidad Pública de Navarra, está pensado para servir de introducción y apoyo práctico a todos aquellos estudiantes de ingeniería y personas en general interesadas en el diseño electrónico mediante la utilización de sistemas microcontroladores de 8 bits. Así, el libro se estructura en diferentes capítulos en los que mediante la realización de ejercicios de diferente complejidad se revisan los módulos que se encuentran generalmente integrados en estos sistemas, como por ejemplo los puertos de entrada/salida digitales, temporizadores, conversores analógicos, comparadores y comunicaciones serie.

Los ejemplos y ejercicios del libro están basados en la arquitectura del PIC16F877A pero se pueden extrapolar fácilmente a otro tipo de microcontroladores gracias al empleo de la programación en lenguaje de alto nivel 'C'. No obstante, las estructuras básicas de programación de este lenguaje se revisan en el primer capítulo del libro para aquellos menos familiarizados con el mismo, mientras que los capítulos siguientes están orientados a la resolución de problemas prácticos.

La metodología empleada permite seguir los ejemplos propuestos a partir de los conocimientos básicos de programación de los primeros capítulos y sin la necesidad de utilizar hardware ni elementos adicionales. Todo ello gracias a la utilización de la herramienta de software gratuita MPLAB IDE®, que integra un simulador, y a la descarga de los materiales necesarios de forma online en la sección de descargas del libro en www.marcombo.com









www.marcombo.com